Learning Gaussisan Noise Models from High-Dimensional Sensor Data with Deep Neural Networks

by

Katherine Y. Liu

B.S., University of California, San Diego (2015)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2018

Author ..... Signature redacted
Department of Aeronautics and Astronautics
February 01, 2018

Certified by ..... Signature redacted
Nicholas Roy
Professor
Thesis Supervisor

Accepted by ..... Signature redacted
Hamsa Balakrishnan
Associate Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

Learning Gaussisan Noise Models from High-Dimensional Sensor Data with Deep Neural Networks

by

Katherine Y. Liu

Submitted to the Department of Aeronautics and Astronautics
on February 1, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

While measurement covariances are often taken to be constant in many robotic state estimation systems, many sensors exhibit different interactions with their environment. Accurate covariance estimation allows graph-based estimation techniques to better optimize state estimates by reasoning about the utility of different methods relative to each other.

This thesis describes a method of learning compact feature representations for real-time covariance estimation. A direct log-likelihood optimization technique is used to train a deep convolutional neural network to predict the covariance matrix of a Gaussian measurement model, given representative data. This method is algorithm-agnostic, and therefore does not require the handcoding of representative features. Quantative results are presented, showing that improved measurement covariances on a frame-to-frame visual odometry system reduce trajectory errors after a loop closure is applied.

Thesis Supervisor: Nicholas Roy
Title: Professor

Acknowledgments

I would like to thank my parents, Susan and Eric, for your support over the years. Thanks to my brother and sister, Max and Lillian, for the phone calls, postcards, and encouragement. Thank you to Bryan, for your endless patience and love. And of course, thank you to Nick Roy, for giving me a chance to be a part of the Robust Robotics Group, and for all of the invaluable guidance. Thanks to all the great people in the Robust Robotics Group for all the good cheer and good advice. Thanks to Kyel Ok for being an awesome research collaborator, and to William Vega-Brown for the foundations of this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

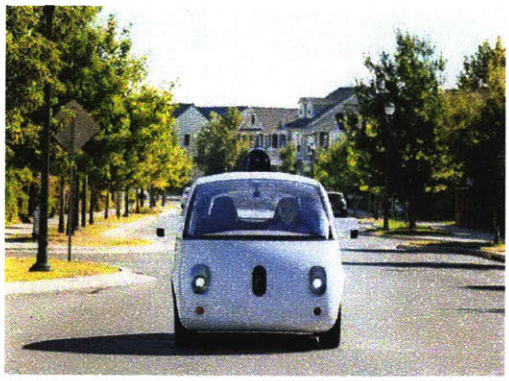
1.1 Motivations for Covariance Estimation

Stochastic State Estimation

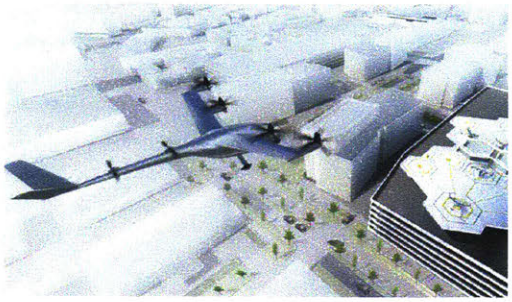
After spending decades in largely artificial and carefully controlled environments, robots are beginning to venture into increasingly realistic conditions. Entire industries are poised to be fundamentally accelerated by autonomy. For example, in the transportation field autonomous cars are rapidly becoming an inevitable reality [13]. Other proposals for consumer-facing autonomous systems include delivery drones and autonomous air taxis [1, 20].

The advent of robots successfully operating in increasingly natural environments can be largely attributed to probabilistic models. Probabilistic frameworks explicitly model the world and interactions with the environment as fundamentally stochastic. These models allow autonomous agents to reason about the world in a manner that takes into account the uncertainty that is present in any action or observation; robustness generally requires reasoning about uncertainty. However, as robots venture into more complicated scenarios (i.e., as the levels of artifice and structure in the environment decrease), they require increasingly complex and potentially hard to specify models of uncertainty.

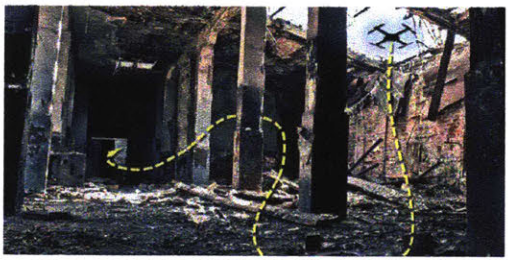
We would like to use size, weight, and power (SWaP) constrained micro-aerial



(a) Waymo's autonmous car [13]



(b) Uber Elevate autonomous air taxi concept [20]



(c) DARPA Fast Lightweight Autonomy (FLA) mission concept [3]

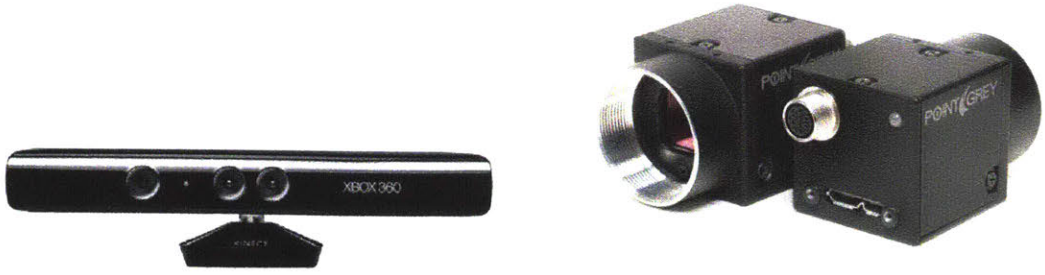
(d) Amazon's delivery drone concept [1] in

Figure 1-1: Examples of robotic products and concepts.

vehicles (MAVs) for a variety of tasks in uncontrived environments without relying on external state estimation frameworks such as Vicon or GPS. Examples of environments without access to GPS include indoors, under the forest canopy, and in disaster zones. In the absence of precise external positioning systems, a MAV must rely on noisy on-board sensors to estimate its own position and orientation. This process, often referred to as *state estimation*, is crucial in any robotic system, as the estimate is then usually consumed by low-level controllers or high level motion planners. In order to properly synthesize noisy measurements from lightweight sensors, stochastic state estimation algorithms require well-specified models of uncertainty.

The Case for Predictive Heteroscedastic Covariances

Sensor measurements are commonly modeled as Gaussian random variables, conditioned upon the state from which the measurement was taken. In practice, the covariance of these random variables is often assumed to be a constant term either



(a) Kinect RGB-D sensor (v1) [2]

(b) PointGrey Flea3 Camera [15]

Figure 1-2: Examples of image-based sensors for robotic state estimation include the Microsoft Kinect RGB-D and the PointGrey Flea3 RGB sensor.

set by hand or derived from empirical data.

Sources of noise may rise from the physical mechanisms of the sensor itself. Consider an inertial measurement unit, which directly measures acceleration and rotational velocity by measuring physical quantities and producing an interpretable electrical signal. Additive noise in these types of sensors is often assumed to be constant, as noise is injected by physical phenomena in the measurement unit itself.

However, assuming constant noise models is not always a safe assumption to make, especially when using more complex sensors that interact with their environments. Furthermore, algorithms are generally used to synthesize raw sensor measurements into lower dimensional measurements of state. We refer to such systems as *sensor-algorithm pairs*, and use the term interchangably with the term sensor, without loss of generality. Sensor-algorithm pairs can exhibit increased noise model complexity, as different types of algorithms may have different shortcomings.

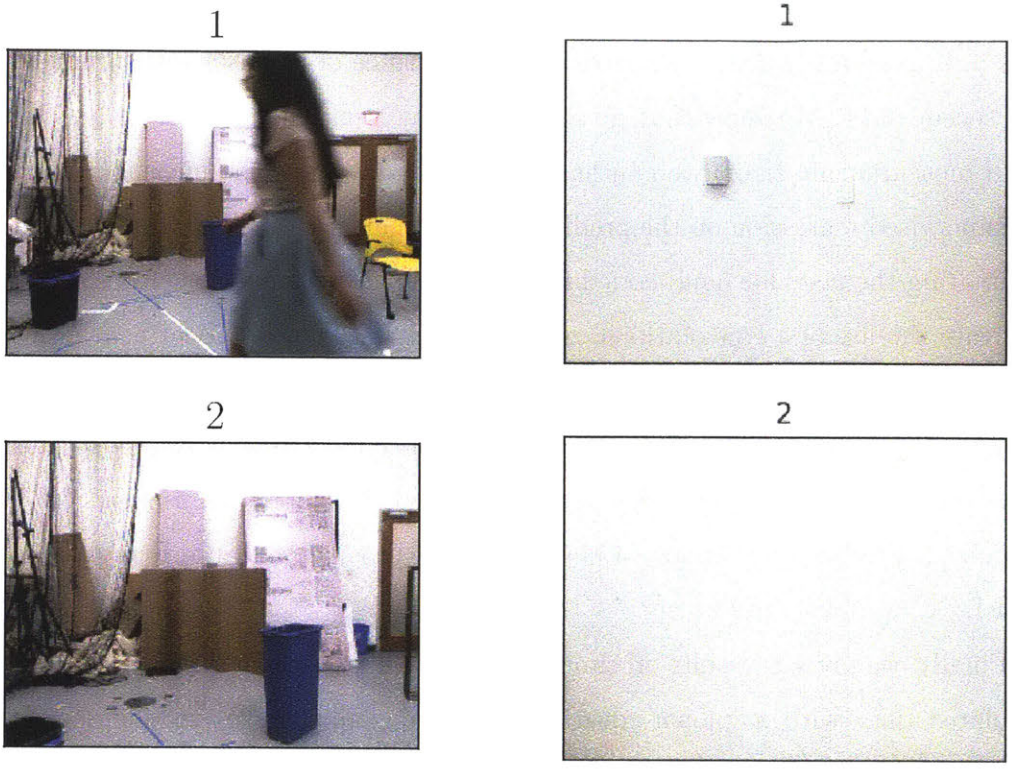
As an illustrative examaple, consider a visual odometry measurement that relies on camera images from image sensors (see Figure 1-2) and a process (such as those presented by Forster et al. 2014, Ok et al. 2016, and Engel et al. 2014) that aligns the images of a static, textured, and constantly-bright environment. One such process might identity a small set of unique features in the image space, and compare the feature locations in a pair of images; the transform between the two image frames can be inferred, given an appropriate camera model [62]. A dense approach to visual

odometry may instead attempt to align every pixel of the image based on an objective function that minimizes photometric error [31].

Consequently, the accuracy of visual odometry measurements in dynamic scenes, low texture environments, and poor lighting conditions is often degraded (the first two scenarios are visualized are show in Figure 1-3). A flat, texture-less wall is an example of a degenerate sensor reading for camera-based systems; without features to properly constrain motion of the sensor, the optimization prcoess in state estimation can be ill-posed. In dynamic scenes, sparse methods may track features on a moving object, which introduces additional error into the triangulation process. Dynamic scenes also introduce noise into dense methods, as the alignment of the pixels is corrupted by non-static object. Furthermore, between dense and sparse methods, there may be different noise characteristics given nuances in the respective optimizations.

The use of poor noise models in state estimation frameworks may have severe consequences, as sensor noise characteristics are fundamental to applications such as the optimal fusion of measurements from several noisy sensors into a single estimate of the vehicle state. If the uncertainty in a measurement is mischaracterized dramatically, the sensor fusion process will incorrectly synthesize measurements by emphasizing high noise measurements. Even on vehicles with a single sensor, noise characteristics are key to evaluating the uncertainty in past estimates and optimally correcting the accumulated error in the presence of additional sources of information, e.g., loop-closure detections added to visual odometry. We are therefore motivated to develop better models of sensor uncertainty to improve the state estimation performance and robustness of SWaP constrained autonomous vehicles.

In this work, we are interested in developing *heteroscedastic* approximations of sensor models that exhibit complex environmental interactions by explicitly considering the uncertainty in the model as a function of some other underlying state. Such models can be difficult to specify as the raw measurements can be very complex and high dimensional.



(a) Example of dynamic vs. static scene (b) Example of textured vs. texture-less scene

Figure 1-3: Examples of sensor images that affect sensor-algorithm measurement qualities. (a) illustrates a dynamic object (a person) which can corrupt the state estimate of an algorithm that assumes a static environment. (b) is an example of a sensor image with very low texture, compared to a scene with no texture.

1.2 Thesis Overview

We present a novel method of measurement covariance estimation that models measurement uncertainty as a function of the measurement itself. We first outline existing work in predictive sensor modeling, covering both homoscedastic and heteroscedastic approaches. We discuss how existing non-parametric and parametric covariance estimation techniques for robotic stochastic state estimation outperform conventional fixed models, but require domain knowledge of the sensors that heavily influence the accuracy and the computational cost of the models. We devote a short chapter to a review of neural network architectures, and how they are optimized.

We then introduce Deep Inference for Covariance Estimation (DICE), which utilizes a deep neural network to predict the covariance of a sensor measurement from raw sensor data. We show that given pairs of raw sensor measurement and groundtruth measurement error, we can learn a representation of the measurement model via supervised regression on the prediction performance of a hyper-parametric model, eliminating the need for hand-coded features and parametric forms. In particular, we motivate the use of a convolutional model for low-level feature identification in complex high dimensional sensor measurements. Our approach is sensor-agnostic, requiring only the measurement errors and raw sensor data. DICE is also constant time; while computation is expensive at train time, once the model parameters have been optimized, predicting is simply a single forward pass through the hyper-parametric model.

Finally, we present results on simulated data and experimental data. We generate simulated data with a known covariance function, allowing us to demonstrate the convergence of the Kullback-Leibler Divergence values between the predicted distribution and groundtruth covariance. We also demonstrate covariance prediction for a visual odometry (VO) system by collecting pairs of RGB-D data and error labels for a specific frame-to-frame VO algorithm. We benchmark DICE predicted covariances against several others in terms of measurement log-likelihood and show that our method better predicts the observed data. Finally, we utilize DICE predicted covariances in a smoothing, graph based trajectory estimation framework, and show that our method results in better positional tracking results than the other methods surveyed.

Chapter 2

Related Work

This chapter outlines several filtering techniques to motivate and ground covariance prediction. A brief overview is then given of existing covariance estimation techniques, including sensor-specific, reactive, and predictive (including non-parametric, parametric, and hyper-parametric techniques).

2.1 State Estimation of Stochastic Systems

We are interested in estimating the state of a robot, relying on noisy sensors to observe the vehicle state. Specifically, we would like to estimate the n-dimensional vehicle state $\boldsymbol{x}_t \in \mathbb{R}^n$ at time t based on a series of p-dimensional measurements or observations, $\boldsymbol{z}_{0:t}$ (where $\boldsymbol{z}_t \in \mathbb{R}^p$). We assume that there exist some deterministic functions $f(x)$ and $h(x)$ which define the motion model and measurement model respectively. In stochastic frameworks, these models are assumed to be corrupted by noise, leading to the following expressions:

$$\boldsymbol{x}_t = f(\boldsymbol{u}_{t-1}, \boldsymbol{x}_{t-1}) + \boldsymbol{v}_t \tag{2.1}$$

$$\boldsymbol{z}_t = h(\boldsymbol{x}_t) + \boldsymbol{w}_t \tag{2.2}$$

where $\boldsymbol{v}$ and $\boldsymbol{w}$ are additive noise samples drawn from the motion and measurement noise models, respectively. A common choice of noise model is a Gaussian distribution,

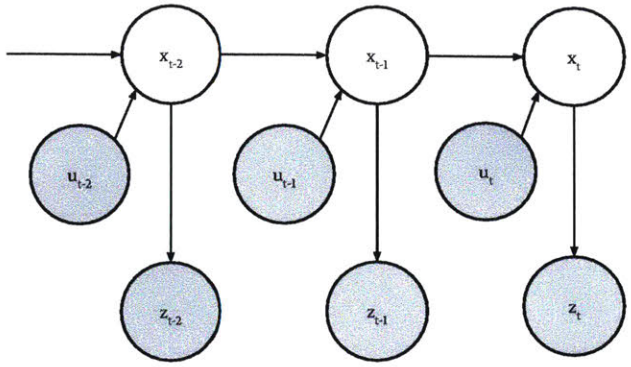


Figure 2-1: Graphical model of the state estimation problem, with the Markov assumption. The white circles indicate hidden variables (the state), and the grey variables are observations (control inputs and measurements). Arrows indicate conditional dependencies.

which is specified by a mean μ and positive definite covariance matrix Γ, i.e., $\mathcal{N}(\mu, \Gamma)$. The probability density function (PDF) of a multivariate Gaussian is as follows:

$$p(x) = |2\pi\Gamma|^{-1/2}\exp(-\frac{1}{2}(x-\mu)^T\Gamma^{-1}(x-\mu)). \tag{2.3}$$

Measurements can be indirect measurements of $\boldsymbol{x}_t$, as in the case of cameras or laser-based range sensors. In these cases, the raw measurements ξ_i are usually very high dimensional, and must be synthesized into lower-dimensional measurements. The measurement model, then, translates raw sensor measurements into measurements of the vehicle state. From this general model, we will now explore both filtering and smoothing techniques for estimating the vehicle state.

2.1.1 Recursive State Estimation

The stochastic state estimation problem can be viewed as finding an estimate for the probability distribution of the current vehicle state, given a series of measurements and control inputs, i.e., $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t})$. As in [71], we can factorize this conditional

distribution using Bayes' rule:

$$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t-1}) = \frac{p(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1})p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1})}{p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1})}. \tag{2.4}$$

However, we observe that this problem becomes quickly intractable as the memory required to maintain the conditional distribution grows exponentially with time. Instead, it is common apply the Markov assumption, which asserts that the state at $t-1$ is a sufficient statistic for the state at t (see Figure 2-1). If we assume that our model is Markovian, we can write the conditional probability of each measurement $\boldsymbol{z}_t$ as only dependent on the previous state $\boldsymbol{x}_t$ (i.e., z_t is *conditionally independent* of the history of control inputs and measurements):

$$p(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1}) = p(\boldsymbol{z}_t|\boldsymbol{x}_t). \tag{2.5}$$

Equation 2.4 can then be re-written as

$$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t-1}) = \eta p(\boldsymbol{z}_t|\boldsymbol{x}_t)p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1}) \tag{2.6}$$

where we have subsumed the marginalization over $\boldsymbol{x}_t$ that was previously in the denominator into η. The second term of Equation 2.6 can be further broken up:

$$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-1}) = \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1})p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1}, \boldsymbol{u}_{0:t-2})d\boldsymbol{x}_{t-1} \tag{2.7}$$

Approaches that exploit the elegant recursive relationship of Equations 2.6 and 2.7 are often called *recursive Bayesian filtering* techniques [71]. The most well-known and popular of this family of techniques is the ubiquitous Kalman filter.

Kalman Filtering

The Kalman filter, introduced by Kalman et al. in 1960, is optimal in the sense that it is an unbiased minimum variance estimator for linear Gaussian systems. Kalman filters make the assumption that the motion and measurement models are linear

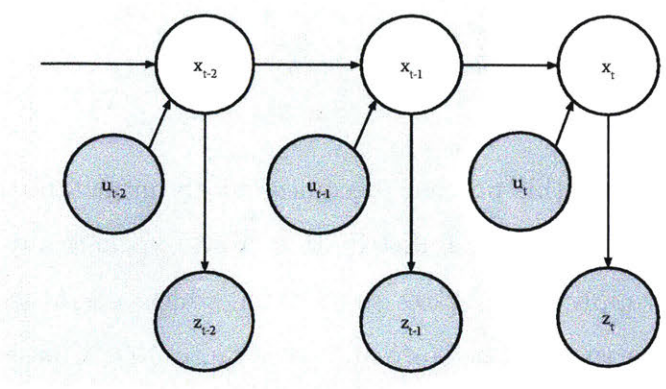


Figure 2-2: In recursive state estimation, all previous measurements and control inputs are marginalized out, and the previous state estimate becomes the prior for the current estimate.

systems, and that the noise models are zero mean Gaussians. Gaussian models are a popular choice of noise model due to their attractive computational characteristics (convolutions and products are reduced to convenient algebraic expressions). Under these assumptions, Equations 2.1 and 2.2 are re-written as:

$$\boldsymbol{x}_{t+1} = F\boldsymbol{x}_t + B\boldsymbol{u}_t + \boldsymbol{v}_t \tag{2.8}$$

$$\boldsymbol{z}_{t+1} = H\boldsymbol{x}_{t+1} + \boldsymbol{w}_t \tag{2.9}$$

where F, B, and H are now constant matrices. Furthermore, the covariance matrices must specified for the Gaussian noise models:

$$\boldsymbol{v}_t \sim \mathcal{N}(0, \boldsymbol{\Omega}_t) \tag{2.10}$$

$$\boldsymbol{w}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}_t) \tag{2.11}$$

and $\boldsymbol{\Sigma}_t \in \mathbb{R}^{p \times p}$. Both covariances must be positive semi-definite. The state $\boldsymbol{x}_t$ is now a normally distributed variable with mean $\boldsymbol{\mu}_t$ and covariance $\boldsymbol{P}_t$. Constraining the estimate of $\boldsymbol{x}_t$ to be unbiased and minimizing the variance of the estimate, the

following equations are derived[1]:

$$\begin{aligned} \bar{\boldsymbol{\mu}}_t &= F_t \boldsymbol{\mu}_{t-1} + B_t \boldsymbol{u}_{t-1} \\ \bar{\boldsymbol{P}}_t &= F_t \boldsymbol{P}_{t-1} F_t^T + \boldsymbol{\Omega}_t \end{aligned} \tag{2.12}$$

$$K_t = \bar{\boldsymbol{P}}_t H_t^T (H_t \bar{\boldsymbol{P}}_t H_t^T + \boldsymbol{\Sigma}_t)^{-1} \tag{2.13}$$

$$\begin{aligned} \boldsymbol{\mu}_t &= \bar{\boldsymbol{\mu}}_t + K_t(\boldsymbol{z}_t - H_t \bar{\boldsymbol{\mu}}_t) \\ \boldsymbol{P}_t &= (I - K_t H_t)\bar{\boldsymbol{P}}_t \end{aligned} \tag{2.14}$$

The equations in 2.12 are sometimes referred to as the *prediction step*, as they simply propagate the previous state estimate through the motion model (Equation 2.1) to predict the distribution of the state should be before any measurements are incorporated. The equations in 2.14 are also known as the *update* or *innovation step*. We direct the reader to Kalman et al. [29] for more detailed derivations.

Extended Kalman Filtering

The Extended Kalman Filter (EKF) is an extension of the Kalman Filter for nonlinear systems. The nonlinear system is linearized by taking a first order Taylor expansion around the current estimate. The motion and measurement Jacobians ($\hat{F}_t$ and $\hat{H}_t$) are defined as

$$\begin{aligned} \hat{F}_t &= \frac{\partial f(\boldsymbol{\mu}_{t-1}, \boldsymbol{u}_t)}{\partial \boldsymbol{\mu}_{t-1}} \\ \hat{H}_t &= \frac{\partial h(\boldsymbol{\mu}_t)}{\partial \boldsymbol{\mu}_t}. \end{aligned} \tag{2.15}$$

The EKF derivation is largely similar to that of the KF. The main difference is that the mean of the previous state estimate is propagated through the nonlinear models, while the covariance of the state estimate is propagated using the linearized models.

[1]Thrun et al. 2005 show how these equations can be derived from Equations 2.6 and 2.7

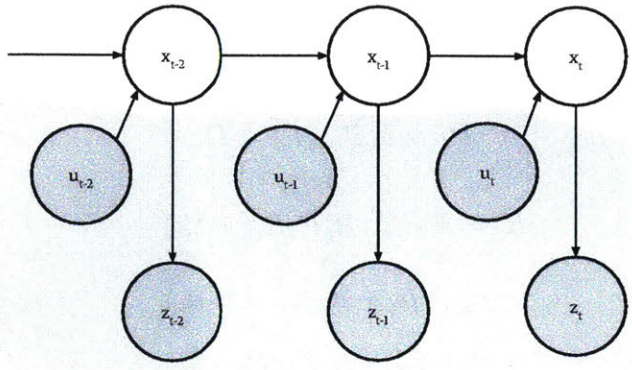


Figure 2-3: Smoothing state estimation model. In the smoothing state estimation problem, the measurements and control inputs are marginalized out, and the solution is an optimal estimate of the entire trajectory.

This is summarized by the following equations:

$$\begin{aligned} \bar{\boldsymbol{\mu}}_t &= f(\boldsymbol{\mu}_{t-1}) + b(\boldsymbol{u}_{t-1}) \\ \bar{\boldsymbol{P}}_t &= \hat{F}_t \boldsymbol{P}_{t-1} \hat{F}_t^T + \boldsymbol{\Omega}_t \end{aligned} \tag{2.16}$$

$$K_t = \bar{\boldsymbol{P}}_t \hat{H}_t^T (\hat{H}_t \bar{\boldsymbol{P}}_t \hat{H}_t^T + \boldsymbol{\Sigma}_t)^{-1} \tag{2.17}$$

$$\begin{aligned} \boldsymbol{\mu}_t &= \bar{\boldsymbol{\mu}}_t + K_t(\boldsymbol{z}_t - \hat{H}_t \bar{\boldsymbol{\mu}}_t) \\ \boldsymbol{P}_t &= (I - K_t \hat{H}_t) \bar{\boldsymbol{P}}_t \end{aligned} \tag{2.18}$$

2.1.2 Smoothing methods

Incremental Smoothing and Mapping

While KFs and EKFs (and other variants of Bayesian recursive estimation) have attractive computational benefits, they can suffer from brittleness due to the repeated marginalization over the previous state. *Smoothing methods* attempt to combat this problem by instead estimating a series of poses. Instead of estimating only $\boldsymbol{x}_t$, smoothing methods estimate $\boldsymbol{x}_{0:t}$ by exploiting the sparsity in the model induced by the Markov assumption. Introduced in 2008 by Kaess et al., Incremental Smoothing and Mapping (iSAM) is one of the most well-known modern smoothing methods [28]. Instead of marginalizing, the iSAM algorithm keeps the entire joint probability, which

can be written compactly as

$$p(\boldsymbol{x}_{0:t}, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}) \approx p(\boldsymbol{x}_0) \prod_{i=1}^{t} p(\boldsymbol{x}_i|\boldsymbol{x}_{i-1}, \boldsymbol{u}_i) \prod_{k=1}^{K} p(\boldsymbol{z}_{i_k}|\boldsymbol{x}_i, l_k). \tag{2.19}$$

where we now also augment the state with the position of K observed landmarks[2] l_k, and $\boldsymbol{z}_{i_k}$ represents the measurement of landmark k at time i (assuming known correspondences). The landmark random variables can be expressed compactly in vector notation, i.e., $\boldsymbol{l}$. The goal of the smoothing problem is to find the *maximum a posteriori* (MAP) estimate of the trajectory and landmark positions, i.e.,

$$\begin{aligned} \boldsymbol{x}^*_{0:t}, \boldsymbol{l}^* &= \underset{\boldsymbol{x}_{0:t}, \boldsymbol{l}}{\operatorname{argmin}}\, p(\boldsymbol{x}_{0:t}, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}) \\ &= \underset{\boldsymbol{x}_{0:t}, \boldsymbol{l}}{\operatorname{argmax}} -\log p(\boldsymbol{x}_{0:t}, \boldsymbol{z}_{0:t}, \boldsymbol{u}_{0:t}), \end{aligned} \tag{2.20}$$

Substituting Equations 2.1, 2.2, and 2.3, we can express 2.20 as a nonlinear least squares problem:

$$\boldsymbol{x}^*_{0:t}, \boldsymbol{l}^* = \underset{\boldsymbol{x}_{0:t}, \boldsymbol{l}}{\operatorname{argmin}} \sum_{i=1}^{t} \|f(\boldsymbol{x}_{i-1}, \boldsymbol{u}_i) - \boldsymbol{x}_i\|^2_{\boldsymbol{\Omega}} + \sum_{k=1}^{K} \|h(\boldsymbol{x}_i, l_k) - \boldsymbol{z}_{i_k}\|^2_{\boldsymbol{\Sigma}} \tag{2.21}$$

where $\|\boldsymbol{e}\|^2_{\Sigma}$ is the Mahalanobis norm that directly scales the modeling error $\boldsymbol{e}$ inversely proportional to the square root of the covariance term Σ, i.e.,

$$\|\boldsymbol{e}\|^2_{\Sigma} \triangleq \boldsymbol{e}^{\mathsf{T}} \Sigma^{-1} \boldsymbol{e} = \|\Sigma^{-\mathsf{T}/2} \boldsymbol{e}\|^2_2. \tag{2.22}$$

Equation 2.21 can be solved using iterative nonlinear optimization techniques such as the Gauss-Newton or Levenberg-Marquardt algorithms.

[2]We observe that although iSAM is presented as a simultaneous localization and mapping (SLAM) algorithm, it can also be used to estimate only the vehicle state, just as the the Kalman filter and variants can be augmented to also estimate the landmark positions.

2.1.3 The Role of Covariances in State Estimation

Covariances play a critical role in both filtering and smoothing state estimation approaches. In Kalman filtering, the noise matrices are used to derive the optimal Kalman gain (Equations 2.13 and 2.17). Not long after the Kalman filter was published, researchers began to explore the role of incorrectly specified noise models in the optimal filtering framework (see the work of Heffes [23] and Nishimura [50]); this work laid the motivations for what later became adaptive Kalman filtering.

When used in smoothing techniques, the covariance matrix generally provides the scaling for the weighted nonlinear least squares objective function, as in Equation 2.21. Intuitively, the Mahalanobis distance penalizes errors from measurements with small covariances more aggressively than those with larger covariances. It is clear that the final solution is sensitive to the correct noise specification.

2.2 Covariance Estimation Techniques

Despite this clear dependence, noise matrices are still most commonly hand-tuned constant values. While this may often work in practice for relatively simple scenarios, as we turn to more complex sensors and measurements, the need to correctly specify all parts of the model is exacerbated. In the next section, we discuss a few of the main bodies of covariance estimation literature.

2.2.1 Sensor or Algorithm Specific Estimation Techniques

One approach to improving sensor noise models is to design algorithms to predict the noise in specific sensors or algorithms. For example, Censi [9] and Barczyk and Bonnabel [6] formulate methods for estimating the covariance of various iterative closest point (ICP) algorithms in a variety of ways. ICP algorithms align sucessive sets of points (such as those generated by laser-based sensors) to find the transformation between measurements. Censi presents a computationally efficient method of covariance estimation based on the curvature of the ICP objective function at the solution,

explicitly considering the algorithm used to estimate the odometry. [9]. The Fisher information matrix is used to determine when the optimization is under-constrained. Barczyk and Bonnabel claim that the discussed Gaussian white noise assumption is inconsistent with observed noise models for ICP algorithms paired with Kinect V1 sensors, and results in severe underestimates of the covariance [6]. Instead, Barczyk and Bonnabel formulate the covariance estimate by explicitly modeling quantization errors. Algorithms formulated for select sensor-algorithm pairs are difficult to generalize or extend because they are developed for very specific systems. The breadth of research itself into creating models for the covariance of specific sensor-algorithm pairs[3] suggests that general purpose covariance estimation techniques will be useful for quickly extending the benefits of covariance estimation to new domains.

The Cramer-Rao bound [10, 58] is also sometimes used as a heuristic for covariance estimates. The Cramer-Rao bound provides a theoretical lower bound on the covariance of an estimator, based on the Hessian of the objective function at the solution. However, because it is only a lower bound, it is often an extreme underestimate of the covariance. Additionally, the guarantees of the Cramer-Rao bound only apply if the estimated solution is truly at the global minimum.

2.2.2 Reactive Estimation Techniques

Early approaches to generalized covariance specification relied heavily on the statistics of the innovation vector. Mehra introduced *adaptive Kalman filtering*, an approach driven by the insight that if the covariance matrices are properly specified, the innovation variable should well modeled as white noise [43]. Mehra presented a two stage algorithm for adapting the covariance. First, a set of innovation updates are tested to see if they are drawn from a white noise distribution, using a 95 percent confidence test. If they do not fit the predicted model, a closed form solution of the new estimate of both Q and R can be calculated, given that the number of measurements is greater than the dimension of the state vector multiplied by the dimension of the measure-

[3]See also Barczyk et al. 2014, Prakhya et al. 2015, etc for more perspectives on ICP covariance estimation.

ment. While theoretically satisfying (closed form solutions for continuous state space models are also presented), this approach still assumes that Q and R are constant, rather than time varying. Adaptive Kalman filtering and other similar approaches are sometimes referred to as *innovation adaptive estimation* (IAE) techniques.

Another online estimation technique is *multiple model adaptive estimation* (MMAE). In MMAE, several Kalman filters with varying noise models are run in parallel, and the state estimate is expressed as a weighted combination of each estimate. Consequently, MMAE methods must assume some prior distribution over weighting parameters, $\boldsymbol{\alpha}$. As in Magill [41], the weighted estimated of the state can be expressed as:

$$\hat{x} = \sum_{i=1}^{L} \hat{\boldsymbol{x}}_k(\alpha_i) P(\alpha_i|\boldsymbol{z}_k), \tag{2.23}$$

where there are L filtering being run in parallel. Given a discrete model, $P(\alpha_i|\boldsymbol{z}_k)$ can be rewritten using Bayes rule

$$P(\alpha_i|\boldsymbol{z}_k) = \frac{p(\boldsymbol{z}_t|\alpha_i)p(\alpha_i)}{\sum_i^L p(\boldsymbol{z}_t|\alpha_i)}. \tag{2.24}$$

So long as a parametric form of $p(\boldsymbol{z}_t|\alpha_i)$ is defined, Equation 2.23 is fully specified. Magill proposed letting $p(\boldsymbol{z}_t|\alpha_i)$ be the a Gaussian distribution defined by the covariance matrix observed by the i-th filter. MMAE can be viewed as an indirect method of covariance estimation; the underlying covariance is implicitly present in the weighting of different models with different noise specifications. Other examples of MMAE include Maybeck and Pogoda [42] and Menke and Maybeck [44].

Mohamed and colleagues benchmarked the utility of several of these reactive techniques in various flavors for a INS/GPS tracking application, noting that noise estimates are important for high accuracy tracking problems [47]. The authors found that parameterizing the covariance matrices in an IAE framework and performing maximum likelihood optimization over the values of a diagonal covariance matrix achieved roughly half the position tracking error (in terms of root mean square error) as compared to a traditional Kalman Filter on their task. For a more recent perspective and overview of reactive estimation techniques, the reader is referred to the

thorough survey in Duník et al. [17].

While the aforementioned general covariance estimation techniques have shown the promise of reasoning more intelligently about the covariance of stochastic systems, they are inherently reactive methods. Several past iterations of the innovation vector must in general be used to calculate a new covariance estimate, which can introduce lag in the system. Furthermore, such methods are useful for state estimation only, and less useful for planning into the future. While some of the techniques are constantly evolving (an improvement from the constant covariance assumption), the statistics are calculated from noisy measurements and therefore often dependent on the calculation window size.

2.2.3 Predictive Estimation Techniques

Covariance estimation techniques that attempt to learn predictive functions present many advantages over reactive techniques. Usually conditioned upon some representation of the state or measurement, predictive covariance estimation techniques discard the assumption of constant covariances entirely and instead attempt to learn the relationship between the observed data and the uncertainty, assuming a heteroscedastic covariance (i.e., that the covariance is a function of some hidden variable, such as the state, or environment).

Non-parametric techniques

Non-parametric estimation techniques are attractive for their model richness and flexibility. Instead of committing to a pre-defined parametric form of the model, non-parametric techniques build relationships based on data. For example, Gaussian processes (GPs) assume that every combination of n variables (i.e., $X_0, X_1, ..., X_2$) is a multivariate Gaussian distribution for any value of n [40]. As a consequence, an estimate for any point in the continuous space of X can be found, along with a covariance for that point, so long as an appropriate kernel function is defined.

Ko and Fox presented a class of methods called GP-BayesFilters, which incorporated the estimates of Gaussian processes into various flavors of Kalman filtering (extended and unscented) and particle filtering [32]. GP-BayesFilters are flexible in that they can leverage sparse GP methods to reduce computational complexity. Given a training set of states x and measurements y, i.e.,

$$\mathcal{D}_{\boldsymbol{x},\boldsymbol{y}} = \left\{ \boldsymbol{x}_i, \boldsymbol{y}_i \quad \forall t \in [1, N] \right\}, \tag{2.25}$$

the GP predicts the mean and variance of the query $\boldsymbol{x}_*$

$$\mathrm{GP}_\mu(\boldsymbol{x}_*, \mathcal{D}_{\boldsymbol{x},\boldsymbol{y}}) = \boldsymbol{k}_*{}^T K^{-1} \boldsymbol{k}_* \tag{2.26}$$

$$\mathrm{GP}_\Sigma(\boldsymbol{x}_*, \mathcal{D}_{\boldsymbol{x},\boldsymbol{y}}) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*{}^T K^{-1} \boldsymbol{k}_* \tag{2.27}$$

where $k()$ is some kernel function, $\boldsymbol{k}_*$ is the kernel distance between the query point and the dataset, and $K \in \mathbb{R}^{N \times N}$ (the kernel matrix of the training data). The hyper-parameters of the kernel function are optimized using a log-likelihood objective function at train time. The approaches were validated experimentally on micro-blimp state estimation task, and the authors showed that the model learns to assign larger measurement covariances when the vehicle motor is on, which causes more aggressive yawing motions, and is therefore causes the vehicle to be harder to track. While modeling the covariance as a function of the state can be useful, if the state itself is an estimate this can lead to complex filter interactions.

Vega-Brown et al. proposed CELLO, a kernel regression technique that estimates the covariance as a function of the measurement itself [73, 72]. CELLO estimates measurement covariances from a database of measurement errors e_i and predictor features ϕ_i

$$\mathcal{D}_{e,\phi} = \left\{ e_i, \phi_i \quad \forall t \in [1, N] \right\}. \tag{2.28}$$

ϕ_i is a low dimensional feature representation of the raw measurements ξ_i, i.e.,

$$\phi(\xi) : \mathbb{R}^q \rightarrow \mathbb{R}^p \tag{2.29}$$

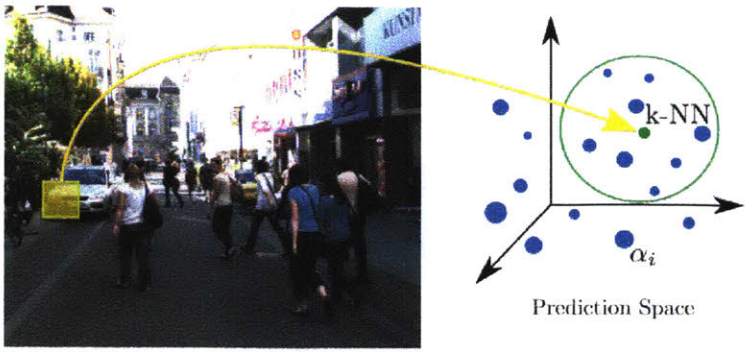


Figure 2-4: Pictorial representation of kernel method for estimating covariance, from [53]. To determine the covariance for a specific patch of pixels, the patch is mapped into a prediction space. The nearest neighbors in that feature space are then used to predict the covariance. Prediction performance is therefore strongly related to the prediction space selected. Figure reproduced with permission of author.

where , $p \ll q$. The authors extend the Nadaraya-Watson estimator [48] to estimate the covariance as an average outer product over $\mathcal{D}_{e,\phi}$:

$$\hat{\Sigma}(\phi;\alpha) = \frac{1}{\sum_{i=1}^{N} k_i} \sum_{i=1}^{N} k_i e_i e_i^T, \tag{2.30}$$

where

$$k_i = k(\rho_\alpha(\phi_i, \phi_0)). \tag{2.31}$$

In Equation 2.31, $k(\cdot)$ is a decreasing weighting function, and $\rho_\alpha(\cdot)$ is some distance function between two predictors, parameterized by α. The parameters of the kernel are then optimized using gradient descent to maximize likelihood:

$$\alpha^* = \underset{\alpha}{\text{argmax}} \sum_{i}^{N} \mathcal{L}(\hat{\Sigma}(\phi;\alpha), e_i). \tag{2.32}$$

Choosing the right predictive feature space is key to the success of the kernel technique; a low dimensional representation that allows the weighting function to choose the appropriately similar samples from the dataset is very important. Intuitively, the covariance estimator presented in Equation 2.30 utilizes the low dimensional feature

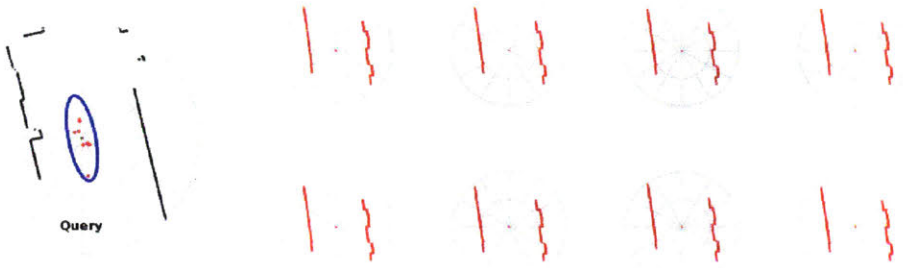


Figure 2-5: Illustration of how a poorly specified feature space may result in poor estimation. On the left, an example of a query laser scan (black) that looks very similar to a hallway, but is constrained in all three dimensions. The error samples associated with the nearest neighbors in the feature space are represented as red dots, and the empirical covariance drawn in blue. To the right, plots of the raw measurements of a few of the nearest scans. Hand-coded features that rely solely on the appearance of the scan can struggle to properly associate the samples in the dataset in these situations; the raw measurements in the defined feature space do not exhibit a back wall.

representation ϕ to find examples in $\mathcal{D}$ that are similar to the query vector ϕ_t. The predictor space allows non-parametric approaches to define a low dimensional manifold that is a function of a reduced set of features. An attractive consequence of such formulations is that dot products (and distances) can be calculated in the newly defined space, even if the relationship between the raw input and the predictors is highly nonlinear. Optimizations are then carried out in the lower-dimensional feature space. Poor feature selection may cause the the kernel approximation to fail to appropriately weight these samples to form an empirical covariance, or fail to properly disambiguate between samples.

One can attempt to manually define the predictive feature space (ϕ) for a single observation using intuitive properties. Ko et al. used very low dimensional pseudo-measurements for the relationships, relying on a carefully hand-designed feature space to allow for efficient estimation, i.e., the parameters of an ellipse fit to the observed ellipse in the external camera frame. Vega-Brown et al. identified informative features of a single laser-scan using the domain knowledge in the failure modes of laser scan-matching algorithms (e.g., degeneracy) and the features that could identify those

modes (e.g., symmetry, no returns, etc). Peretroukhin et al. expanded this success to cameras by defining the features that represent the quality of a visual observation as a set of visual-inertial properties such as angular velocity, local image entropy, blur, optical flow variance score, etc [54].

This approach, however, is fundamentally limited by the system designer's ability to identify the best set of features. In practical terms, it requires an iterative improvement process by which features are added, removed, and altered, until the performance of the filter is acceptable. In the case of CELLO, this is illustrated by Figure 2-5, which shows the samples in an example dataset that are closest in the defined feature space to the query laserscan (shown on the left). In ICP-based approaches to laser scan-matching, the optimization can be poorly constrained in directions without structure[4], and it is therefore extremely useful to correctly predict when such poorly constrained optimizations will result in large measurement errors (i.e., large covariances). While the query scan is a fully constrained environment, the representation can group such a measurement with less constrained hallways (missing the back hallway), when using a suboptimal set of hand-coded features. This suggests that the feature representation may not be a sufficient statistic for the prediction task. Ideally, we would like to be able to instead learn this low dimensional feature space from representative data, and optimize the predictor representation itself.

Therefore, while non-parametric techniques have attractive modeling properties (especially regarding the model-richness they support), they often require experts with domain knowledge to hand-code a feature set for computational purposes. In addition, non-parametric approaches tend to scale poorly with the size of the dataset. While sparse Gaussian process methods do exist, they tend to decrease computation by reducing the number of data points used for estimates (see Herbrich et al. [24], Quiñonero-Candela and Rasmussen [57]); for these approaches to succeed, the chosen feature space must still adequately express the raw relationship to the quantity being

[4]For an extreme case of such a phenomena, consider the *infinite corridor* problem, with is an example of degenerate environment for the laser-ICP sensor-algorithm pair. If a planar LIDAR with a limited range and field of view sees two successive scans that are each only two parallel lines (each wall of the hallway, with the end of the hallway is out of sight), then any objective function that attempts to align the two measurements will have and infinite number of solutions.

estimated.

Parametric techniques

In contrast to non-parametric approaches, heteroscedastic parametric approaches do assume a specified form of the function estimating the covariance matrix. Hu and Kantor proposed a parametric covariance estimation technique for heteroscedastic noise, in which a linear parametric model is assumed, such that

$$\hat{\Sigma}(\xi_i) = M(\phi(\xi_i); \alpha) \tag{2.33}$$

and

$$M(\phi(\xi_i); \alpha) : \mathbb{R}^m \rightarrow S_{++}^{nxn}, \tag{2.34}$$

where M is some function that maps a vector of predictor features to a positive definite matrix [27]. In this framework, α (the output of the linear parametric model) is used to construct an LDL^T matrix (a modified Cholesky decomposition). The optimal parameters α^* are then found by minimizing the negative log-likelihood over a training dataset as in Equation 2.28:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \sum_{i}^{N} \mathcal{L}(M(\phi(\xi_i); \alpha), e_i). \tag{2.35}$$

Like CELLO, the algorithm presented by Hu et al. assumes the existence of a low dimensional feature representation; we have already discussed the difficulties of this deceptively simple assumption. Parametric techniques tend to require even more expert handling than non-parametric, as a good structure and form of the function must be assumed - in this case the authors use a simple linear model. If a good model exists *a priori*, parametric methods can be quite efficient and compact. For complex heteroscedastic relationships, such models can be difficult to specify.

Hyper-parametric techniques

We distinguish hyper-parametric function approximation techniques (i.e., deep learning) from non-parametric and parametric by their often over-parameterized models[5]. Where non-parametric models gain their structure from the data itself, hyper-parametric methods provide models with many parameters that are then optimized for the task specified by the objective function. Neural networks are particularly attractive for their potential model richness.[6]. For the remainder of this work, we will use the terms *neural networks* and *hyper-parametric methods* interchangeably.

Unlike some of the non-parametric models discussed in early sections, the general purpose architectures of neural networks do not immediately provide closed form solutions for well-formed probability density functions, which is required for any noise model to be incorporated into a filtering or inference framework. One approach is to estimate a probability for any input value of a joint (or conditional) probability. Modha and Fainman [46] derived a loss function for general density estimation using neural networks, seeking to find an expression for the joint $p(X, W)$, where X are the input variables, and W the bounded weights of a single layer deep neural network f, i.e.,

$$p(X, W) = F(c(W) + f(x, W)), \tag{2.36}$$

and F is chosen to be an exponential function for easy log-likelihood computation. Because an explicit parametric form is not assumed, some care must be taken to ensure that the probability function is proper: a normalization value $c(W)$ must be calculated to ensure that the probability density function is always positive and integrates to one:

$$p(X, W) \geq 0 \tag{2.37}$$

$$\int_S p(X, W) dX = 1. \tag{2.38}$$

The requirements in Equations 2.37 and 2.38 can be computationally expensive to

[5]We observe that single layer perceptron models are themselves essentially simple parametric models when using hand-coded features to reduce the dimensionality of the input space.

[6]See Chapter 3 for a more detailed overview of neural network models.

enforce; the authors suggest using forms for which there are closed form solutions to the marginalization, or using Gibbs sampling.

Sarajedini et al. extended learning the joint density function to learning the conditional PDF $p(y|x)$, which is also estimated without assuming a parametric form of the PDF [61]. This is achieved by factoring the conditional probability as $p(y|x) = p(x,y)/p(x)$. The authors derived a loss function for general density estimation using an exponential function for easy log-likelihood computation, utilizing hidden layers with logarithm activation functions and output layers with exponential activations, and also impose bounds on the network weights. As in Modha and Fainman [46], the calculation of the gradients requires potentially expensive integral calculations; the authors suggested methods such as Monte Carlo integration.

While learning arbitrary noise distributions is interesting, in practice many of the efficient filtering algorithms at our disposal assume Gaussian distributions. Perhaps the most popular family of filtering algorithm that can accommodate non-Gaussian distributions is particle filtering (see Gordon et al. [22] and Doucet et al. [16] for examples), but these techniques tend to be computationally expensive, as many state estimators are run in parallel and enough samples must be generated to reflect the underlying distribution. Additionally, the computational burden of enforcing proper probability distributions for any arbitrary distribution can be non-trivial. In this work, we focus on predicting Gaussian measurement noise distributions as a function of high dimensional raw sensor measurements for use in robotic state estimation.

Williams described how neural networks could be used to model conditional multivariate densities [75, 76], circumventing the difficulty in ensuring properly specified PDFs by constraining the estimation to Gaussian distributions. The objective function of the optimization is also to minimize the log-likelihood over the data:

$$E = \sum_{p=1}^{N} E_p \tag{2.39}$$

$$E_p = \frac{1}{2}\log|\Sigma_p| + \frac{1}{2}(y_p - \mu_p)^T \Sigma_p^{-1}(y_p - \mu_p). \tag{2.40}$$

The authors propose special "dispersion" nodes to learn the parameters of a Cholesky decomposition. They demonstrate the approach on one and two dimensional examples, and extend to time series financial data. In addition to the raw time series data, the authors also add hand-coded features to the input vector to identify the periodicity.

In practice, many examples of neural networks being used to estimate conditional distributions still rely on some degree of expert feature extraction. In Cawley et al. [8] and Nunnari et al. [51], the authors presented case studies of neural network techniques for various environmental prediction tasks; for some of the more complex time-series data, the inputs to the network were hand-chosen from a variety of sensor measurements. Avanaki et al. [4] proposed using a neural network to predict the noise parameters of a Rayleigh distribution modelling the speckling in optical coherence tomography by providing a neural network with extracted features from the image such as kurtosis. The reliance on a hand-specified feature space is problematic as we have discussed previously, and generally dilutes the promise of neural networks as flexible feature-learning frameworks.

In the 1990s, neural networks largely fell out of favor for various reasons[7]. In recent years, they have experienced a resurgence in popularity due to increased availability of computation power and data[8]. Now that deep neural networks are affordable to train, they are being exploited for increasingly high dimensional and complex tasks, ranging from image and video classification [35, 33, 30], to video game playing [45, 64]. We believe that data-driven feature discovery presents an exciting opportunity to develop noise models that do not require hand selecting low dimensional feature manifolds or careful data pre-processing.

In this work, we are interested in learning Gaussian noise models based on high dimensional sensor data, and exploit recent advances in hardware to utilize more complex network structures and implicitly learn low-dimensional feature representations.

[7]See Goodfellow et al. 2016 for a review of the history of neural networks.

[8]Sarajedini et al., provided a useful historical perspective in [61]: the authors noted that kernel methods for conditional probability estimation were more common at the time due to the computational costs of training neural networks, reporting that a neural network with 60 hidden layers took over a day to train.

Instead of specifying a hand-coded features space, we use a deep convolutional neural network as a function approximator to implicitly perform feature extraction.

Chapter 3

Neural Network Models

In this chapter, we briefly cover neural network fundamentals. We first describe how network layers are specified, then discuss how backpropagation is used to optimize the free parameters of the model.

3.1 Hyperparametric Function Approximation

Motivated by theories of neurological processes, neural networks approximate functions through the connection of *perceptrons*[1]. Fundamental to neural networks is the notion of information passing; the outgoing signal of a single perceptron is based on a linear combination of the incoming signals to that perceptron. This simple parameterization is extremely powerful: Cybenko et al. showed in their seminal 1989 work [11] that a single layer perceptron of finite width can approximate continuous functions under relatively mild conditions. The *universal approximation theorem* was later strengthened for various combinations of networks and activation functions (such as in Hornik et al. [26], Kŭrková [34], and Leshno et al. [37]). Neural networks are specified by neurons composed into *layers*. Two popular layer types are *fully connected layers* and *convolutional layers*.

[1]We will use the terms perceptrons, neurons, and nodes interchangeably here.

3.1.1 Fully Connected Layers

We begin by describing the fully connected layers of the network.

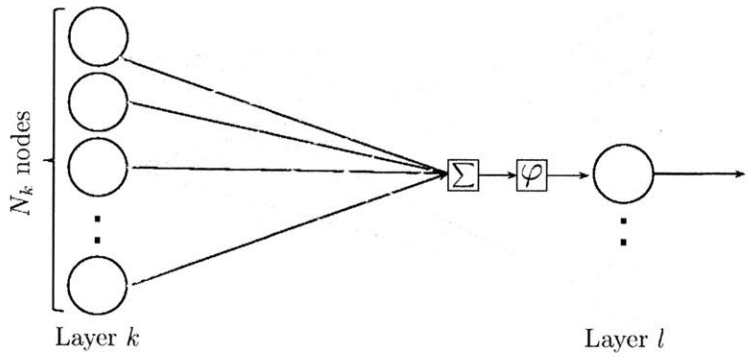


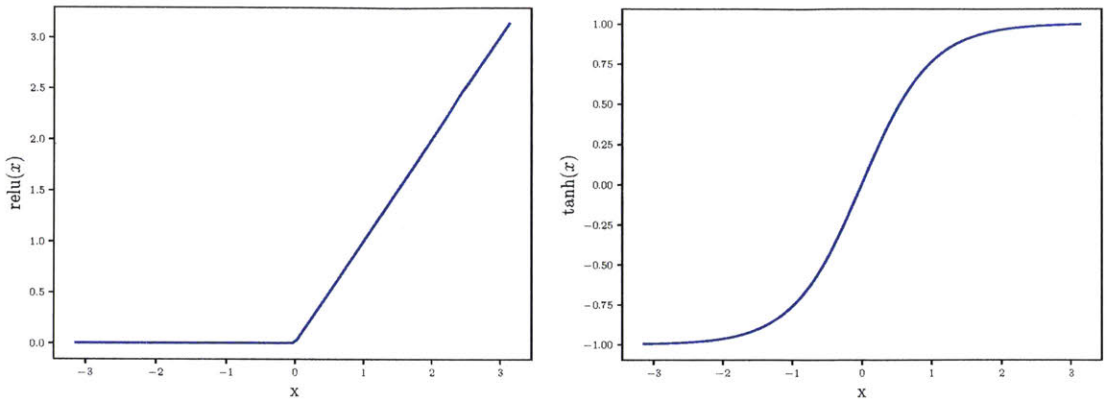
Figure 3-1: An illustration of how the activation of a single node in the l-th layer of a fully connected perceptron is calculated as the weighted sum of the nodes of the previous layer, passed through a non-linear activation function.

In this work, we specifically consider a set of fully connected layers to be *feed-forward neural networks*, where each layer l has incoming connections from layer k (where generally $k = l-1$) and outgoing connections to the next connected layer. We denote the activation of the i-th neuron in layer l to be o_{l_i}, and express it as weighted sum of all the neurons (indexed by N_k) in the previous layer, as in Russell et al. [60] i.e,:

$$o_{l_i} = \varphi\Big(\sum_{k_j \in N_k} w_{k_j,l_i} o_{k_j} + b_{l_i}\Big), \tag{3.1}$$

where w_{k_j,l_i} is the weight value unique to each connection between two nodes o_{k_j}, o_{l_i}, and b_{l_i} a bias term. The weighted sum is then passed through a differentiable non-linear activation function, φ. Popular activation functions include hyperbolic tangent (tanh) and rectified linear units (relu), which are pictured in Figure 3-2. Figure 3-1 visualizes the direction of information flow from one layer to a single node in the next layer; each edge in the network represents a weighted gain.

Stacking perceptron layers yields a *multi-layer perceptron* (MLP). Denote the input to the MLP Υ. Equation 3.1 can also be expressed in vector notation for



(a) Rectified linear activation function. (b) Hyperbolic tangent activation function.

Figure 3-2: Examples of non-linear activation functions.

compactness, such that:

$$\boldsymbol{o}_l = \varphi(\Psi_{lk}^{\Sigma}), \tag{3.2}$$

and

$$\Psi_{lk}^{\Sigma} = \boldsymbol{W}_{lk}\boldsymbol{o_k} + \boldsymbol{b}_{lk}, \tag{3.3}$$

where $\boldsymbol{o}_l$ is now simply the concatenation of all the N_l individual neurons o_{l_i}, $l_i \in N_l$ in layer l, $\boldsymbol{b}_k \in \mathbb{R}^{N_{l-1}}$, and $\boldsymbol{W}_{lk} \in \mathbb{R}^{N_l \times N_{l-1}}$. If we have m fully connected layers, then we denote the network as a function $f_m : \mathbb{R}^{N_\Upsilon} \rightarrow \mathbb{R}^{N_m}$

$$f_{\mathrm{m}}(\Upsilon) = \varphi(\boldsymbol{W}_m\varphi(\ldots\varphi(\boldsymbol{W}_1\vec{\Upsilon} + \boldsymbol{b}_1)\ldots) + \boldsymbol{b}_k), \tag{3.4}$$

where we have renumbered the weight matrices and bias vectors sequentially for readability. The number of nodes in each layer is a design parameter; N_m is set to have equal dimensions as the quantity being approximated. The model f_m can then be optimized through the setting of the variables $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_m$ and $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m$.

3.1.2 Convolutional Layers

In contrast to fully connected layers, convolutional layers explicitly model local connections in the feature space. At each convolutional layer m, $k \in \mathbb{N}$ convolutional

filters $\boldsymbol{W}_{m_k}$ are defined. We introduce here the concept of three-dimensional outputs $o \in \mathbb{R}^{d_1 \times d_2 \times c_k}$, which can be thought of as two-dimensional signals $o^c \in \mathbb{R}^{d_1 \times d_2}$ (indexed by i, j) with $c \in c_k$ channels, or *feature maps*. RGB images are a common example of a three dimensional output o. In this case, i, j index into pixel locations, and c into the color channel (e.g., the red color channel)[2].

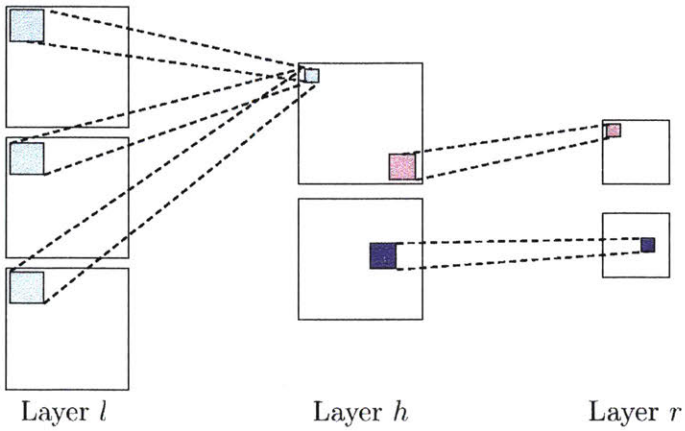


Figure 3-3: A simple diagram of information flow through convolutional layers in a neural network. The value of a pixel in layer h_k (where k is the filter number) is the sum of the convolutional response over all channels of layer l at the query location. Layers h and r show the information flow for maxpool layers; each patch of pixels in layer h is downsampled to a single pixel in layer r.

Convolutional layers have been interpreted as feature discovery layers [36], [77]. Perhaps the simplest convolutional filter is that of a line detector; a convolutional layer with a "horizontal-line" filter will create a "feature map" indicating where such horizontal lines exist as defined by the convolutional response (see Figure 3-4). In contrast to fully connected models, convolutional models share weights (i.e., each channel of an incoming image is passed through the same filter); this allows the layers to aggregate macro-level features across several channels.

As in Goodfellow et al. [21], we begin by defining the convolution operator $*$ of

[2]We will sometimes refer to these three dimensional outputs o as "images", and their features $o_i^c j$ as "pixels" in an effort to distinguish these signals from those used in the perceptron model developed previously.

an input o_{ij} with a filter $\boldsymbol{W_k} \in \mathbb{R}^{q_k \times p_k}$ as

$$\boldsymbol{W_k} * o^c_{ij} = \sum_{q \in q_k} \sum_{p \in p_k} o^c_{q+i,p+j} w_{q,p}, \tag{3.5}$$

where $w_{q,p}$ indexes into $\boldsymbol{W_k}$; this is visualized conceptually in Figure 3-3. The value of a single coordinate value o_{ij} of the k-th feature map in layer m is the result of summing the convolutional responses over each channel l of the set of feature maps of previous layer L, i.e.,

$$\begin{aligned} \Psi^*_{m_k}(i,j) &= \sum_{l \in L} \boldsymbol{W}_{m_k} * o^l_{ij} + b_{m_k} \\ o^{m_k}_{ij} &= \varphi(\Psi^*_{m_k}(i,j)) \end{aligned} \tag{3.6}$$

where $\boldsymbol{W}_{m_k} \in \mathbb{R}^{d_1 \times d_2}$ is the filter to be convolved with the corresponding patch o_{ij}, and b_{m_k} a bias value (φ is again a non-linear activation function). Figure 3-3 illustrates how the value of a neuron at a particular location depends on patches in the channels of the previous layer[3]. We compactly express a single feature layer activation as

$$\begin{aligned} o^{m_k} &= \varphi(\sum_{l \in L} \boldsymbol{W}_{m_k} * o^l + \boldsymbol{b}_{m_k}) \\ &= \varphi(\boldsymbol{W}_{m_k} \odot o + \boldsymbol{b}_{m_k}). \end{aligned} \tag{3.7}$$

For convenience we define the $\odot$ operator between filter $\boldsymbol{W}_a$ and input o with B channels as

$$\boldsymbol{W}_a \odot o = \sum_{b \in B} \boldsymbol{W}_a * o^b. \tag{3.8}$$

We then concatenate the feature maps given by each convolutional filter for the response of the m-th layer, assuming η convolutional filters are specified:

$$o^m(\boldsymbol{\xi}) = [o^{m_0}(\boldsymbol{\xi}), \ldots, o^{m_\eta}(\boldsymbol{\xi})]. \tag{3.9}$$

In deep convolutional networks, convolutional layers are chained together; the feature

[3]We note here that we have neglected to discuss topics such as stride, offset, and zero padding as implementation details.

maps in o^m are then interpreted as channels for the $m+1$ layer, for which another set of filters is specified. We assume here that in our architecture convolutional layers are followed by maxpool layers to reduce the size of the input to the next convolutional layer and introduce some measure of input invariance. The maxpool operator maps patches of pixels into a single value, such that $\mathrm{M} : \mathbb{R}^{n \times m} \rightarrow \mathbb{R}^1$, and can be defined as

$$\mathrm{m}(o^c_{ij \in P}) = \max(P) \tag{3.10}$$

where P is the set of all pixels being pooled. The maxpool operator downsamples the image o^c by keeping only the largest pixel value.

Given an input signal $\boldsymbol{\xi}$, we define the mapping $f_k(\boldsymbol{\xi}; \boldsymbol{W}_1, ..., \boldsymbol{W}_k, \boldsymbol{b}_1, ..., \boldsymbol{b}_k) \mapsto \mathbb{R}^r$, where

$$f_k(\boldsymbol{\xi}) = \varphi(\boldsymbol{W}_k \odot [...(...([\varphi(\boldsymbol{W}_1 \odot \boldsymbol{\xi}) + \boldsymbol{b}_1), ...]...)...] + \boldsymbol{b}_k) \tag{3.11}$$

Figure 3-4: A very simple illustrative example of the post-convolutional feature maps. In this example, the first convolutional filter is a 3x3 matrix of zeros with a single column of ones. The second filter is a 3x3 matrix of zeros with a single row of ones. The resulting feature maps after passing the convolutional

3.2 Optimization

Supervised training of a neural network representation involves finding values of the weight and bias matrices such that some objective function is minimized, given a

training dataset of V input and label pairs $\mathcal{D} = \{\boldsymbol{X}_j, \boldsymbol{Y}_j \; \forall \; j \in [1, V]\}$, i.e.,

$$\boldsymbol{W}^*_{1:k}, \boldsymbol{b}^*_{1:k} = \underset{\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}}{\operatorname{argmin}} \frac{1}{V} \sum_{j=1}^{V} J(\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}; y_j) \tag{3.12}$$

where the objective J is a function of the parameters of the network and the training dataset (examples include mean squared error, logistic loss, etc).

3.2.1 Mini-batch Stochastic Gradient Descent

Note that Equation 3.12 is a nonlinear optimization problem, and we can pose gradient steps as

$$\boldsymbol{P}^a = \boldsymbol{P}^{a-1} + \beta \sum_{j=1}^{V} \frac{\mathrm{d}J}{\mathrm{d}\boldsymbol{P}} \tag{3.13}$$

where we have, in a slight abuse of notation, subsumed all the parameters $\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}$ into a single vector $\boldsymbol{P}$, and $\boldsymbol{P}^a$ is the value of $\boldsymbol{P}$ after a steps of gradient descent.

Due to the computational constraints of readily available computing resources, it is often infeasible to process all of the training data at once. Stochastic Gradient Descent (SGD) randomly samples the training dataset, and takes one gradient step using each sampled training datum (refer to Rosenblatt [59] for an early example of SGD used in neural networks). Mini-batch SGD is a method of iteratively choosing a random subset of v training samples from $\mathcal{D}$, and creating a smaller, pseudo dataset of indices $V_{mini} \in \mathbb{R}^v$. The optimization then becomes

$$\boldsymbol{W}^*_{1:k}, \boldsymbol{b}^*_{1:k} = \underset{\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}}{\operatorname{argmin}} \frac{1}{v} \sum_{j \in V_{mini}} J(\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}; y_j). \tag{3.14}$$

See Goodfellow et al. [21] for more details on different variants of stochastic gradient descent.

3.2.2 Backpropagation

To optimize the objective function given in Equation 3.13, we rely on the backpropagation technique for neural network optimization. Observe that in order to

evaluate any variation of Equation 3.13, we must be able to individually calculate $\frac{\mathrm{d}J}{\mathrm{d}w_{ij}}$, i.e., the gradient step for each individual weight parameter in the network. Back-propagation as an algorithm involves two phases: forward-propagation to determine the output of the network and back-propagation to calculate the gradient at each weight with respect to the objective function.

We first describe here the process of backpropagation for an arbitrary weight w_{ij} in a fully connected network. As in Russell et al. [60], we can show that the backpropagation equations are simply a consequence of sucessive applications of the chain rule. We denote the objective function J, and use the chain rule to express the gradient as a product of partial derivatives:

$$\frac{\mathrm{d}J}{\mathrm{d}w_{ij}} = \frac{\mathrm{d}J}{\mathrm{d}\Psi_j^\Sigma}\frac{\mathrm{d}\Psi_j^\Sigma}{\mathrm{d}w_{ij}}. \tag{3.15}$$

If the node o_j is an output node, $\frac{\mathrm{d}J}{\mathrm{d}\Psi_j^\Sigma}$ is simple to calculate, as it is simply the partial derivative of the objective funtion with respect to the output node value, multiplied by the partial derivative of the output node value with respect to Ψ_j^Σ i.e.,

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}\Psi_j^\Sigma} &= \frac{\mathrm{d}J}{\mathrm{d}o_j}\frac{\mathrm{d}o_j}{\mathrm{d}\Psi_j^\Sigma} \\ &= \frac{\mathrm{d}J}{\mathrm{d}o_j}\frac{\mathrm{d}}{\mathrm{d}\Psi_j^\Sigma}\varphi(\Psi_j^\Sigma).\end{aligned} \tag{3.16}$$

If the node is a hidden node, then intutively we must determine what portion of the error signal is a result of Ψ_j^Σ. The term $\frac{\mathrm{d}J}{\mathrm{d}\Psi_j}$ can be further broken up into the sum over the derivative inputs, ie.,

$$\begin{aligned}\frac{\mathrm{d}J}{\mathrm{d}\Psi_j^\Sigma} &= \sum_{l\in L}\frac{\mathrm{d}J}{\mathrm{d}\Psi_l^\Sigma}\frac{\mathrm{d}\Psi_l^\Sigma}{\mathrm{d}\Psi_j^\Sigma} \\ &= \sum_{l\in L}\frac{\mathrm{d}J}{\mathrm{d}\Psi_l^\Sigma}\frac{\mathrm{d}\Psi_l^\Sigma}{\mathrm{d}o_j}\frac{\mathrm{d}o_j}{\mathrm{d}\Psi_j^\Sigma} \\ &= \sum_{l\in L}\frac{\mathrm{d}J}{\mathrm{d}\Psi_l^\Sigma}w_{jl}\frac{\mathrm{d}}{\mathrm{d}\Psi_j}\varphi(\Psi_j^\Sigma),\end{aligned} \tag{3.17}$$

where L is the set of indices such that o_l receives input from o_j.

The second term in Equation 3.15 is the the partial derivative of the function governing the combination of inputs to o_j, with respect to the weight w_{ij}. For the fully connected layers, it is easy to see that this is simply o_i:

$$\begin{aligned}\frac{\mathrm{d}\Psi_{ij}^{\Sigma}}{\mathrm{d}w_{ij}} &= \frac{\mathrm{d}}{\mathrm{d}w_{ij}} \sum_{q \in N_{j-1}} w_{qj} o_q + b_j \\ &= o_i,\end{aligned} \tag{3.18}$$

where N_{j-1} is the set of all inbound edges to node o_j. Equations 3.17 and 3.18 allow us to recursively compute the gradient defined in Equation 3.15.

The derivation of $\frac{\mathrm{d}\Psi}{\mathrm{d}w_{ij}}$ for convolutional networks is very similar, except that Ψ is instead given by the convolution operator. Let us refer generally to a weight in a convolutional filter k between two arbitrary nodes as w_{xy}; we then have:

$$\frac{\mathrm{d}\Psi_*}{\mathrm{d}w_{xy}} = \frac{\mathrm{d}}{\mathrm{d}w_{xy}} \sum_{q \in q_k} \sum_{p \in p_k} o^c_{q+i,p+j} w_{q,p} \tag{3.19}$$

and

$$\frac{\mathrm{d}\Psi_*}{\mathrm{d}o_{xy}} = \frac{\mathrm{d}}{\mathrm{d}o_{ij}} \sum_{q \in q_k} \sum_{p \in p_k} o^c_{q+i,p+j} w_{q,p}, \tag{3.20}$$

where the indices for which the convolution is valid are determined by the filter and input channels.

The calculation of the derivatives $\frac{\mathrm{d}\Psi_*}{\mathrm{d}w_{ij}}$ and $\frac{\mathrm{d}\Psi_*}{\mathrm{d}o_{xy}}$ allow error information to be propagated through the convolutional layers (using the chain rule), just as in the approach taken for fully connected layers (see Equation 3.17). The calculation of the derivative of the objective function with respect to the bias vectors can be again calculated using the same techniques. Modern neural network libraries generally use computation graphs to handle the derivative information passing; for an in-depth discussion of modern computation graphs for propagating gradient information in convolutional neural networks, as well as how to calculate the gradient for different types of convolution, we refer the reader to Goodfellow et al. [21].

The backpropagation algorithm, then, is a recursive algorithm that must be run from the output of the network J to the input; after $\frac{\mathrm{d}o_l}{\mathrm{d}\Psi}$ is calculated for a layer, it

can be used to calculate $\frac{dJ}{do_{l-1}}$ for the preceding layer.

Chapter 4

Learning Covariances from Data

In this chapter, we present a method of using a deep neural network to estimate the covariance matrix of a complex sensor-algorithm pair, assuming a multivariate Gaussian noise model. First, we pose covariance estimation as a maximum likelihood estimation problem over a training dataset of sensor measurements and associated error measurements. We then describe how the uncertainty prediction is modeled with a neural network, using a negative-loglikelihood objective function and a modified Cholesky decomposition to ensure positive definiteness.

4.1 Preliminaries

We consider a robot with state $\boldsymbol{x}_t \in \mathbb{R}^n$, equipped with a sensor providing a direct measurement $\boldsymbol{z}_t \in \mathbb{R}^p$, where $p \leq n$, at each time step t. We assume that the measurement process may consist of an initial stage to obtain a high dimensional raw measurement $\boldsymbol{\xi}_t \in \mathbb{R}^m$ and a follow-up stage to process the raw measurement into a direct measurement $\boldsymbol{z}_t$ of the state. Therefore, the measurement function could be a function of both $\boldsymbol{\xi}_t$ and $\boldsymbol{x}_t$, i.e., $h(\boldsymbol{x}_t, \boldsymbol{\xi}_t)$. For example, consider a GPS sensor, which first obtains a raw measurement $\boldsymbol{\xi}_t$ of time-of-flight to satellites, then processes it via multilateration to obtain a direct measurement $\boldsymbol{z}_t$ of the sensor position $\boldsymbol{x}_t$.

We assume Gaussian distributions for the conditional probabilities of observation $p(\boldsymbol{z}_t|\boldsymbol{x}_t; \boldsymbol{\xi}_t, \bullet_t)$, where $\bullet_t$ is a set of known variables specific to the measurement model.

Given appropriate functions for expected observation $h(\boldsymbol{x}_t, \boldsymbol{\xi}_t, \bullet_t)$ along with their uncertainties in the form of positive definite covariance matrices $\boldsymbol{\Sigma}_t \in \mathbb{R}^{p \times p}$, we can write

$$\boldsymbol{z}_t \sim \mathcal{N}(h(\boldsymbol{x}_t, \boldsymbol{\xi}_t, \bullet_t), \boldsymbol{\Sigma}_t). \tag{4.1}$$

4.2 Approach

4.2.1 Problem Formulation

We would like to predict the distribution over sensor measurement error $\boldsymbol{e}_t \in \mathbb{R}^p$ at time t, conditioned on the raw measurement $\boldsymbol{\xi}_t$, i.e., predict the distribution $p(\boldsymbol{e}_t|\boldsymbol{\xi}_t)$. We define

$$\boldsymbol{e}_t = |\boldsymbol{z}_t^* - \boldsymbol{z}_t|, \tag{4.2}$$

where $\boldsymbol{z}_t^*$ is the groundtruth measurement, and $\boldsymbol{z}_t$ the observed measurement.

Assuming the distribution to be a zero-mean Gaussian (assuming an un-biased sensor), we focus on predicting the covariance $\boldsymbol{\Sigma}_t$ of the distribution as a function of the raw measurement $\boldsymbol{\xi}_t$, i.e., forming the predictive function g where

$$\boldsymbol{\Sigma}_t \approx g(\boldsymbol{\xi}_t). \tag{4.3}$$

We choose to model the noise as a function of the raw measurement[1]. The approximator function g must capture the highly complex mapping between the raw sensor data to the covariance matrix $\boldsymbol{\Sigma}_i$. One way to predict a complex mapping is by over-parameterizing the approximator function, then learning the parameters using a labeled training dataset $\mathcal{D}$. If an ideal training dataset $\mathcal{D} = \{\boldsymbol{\xi}_i, \mathcal{N}(0, \boldsymbol{\Sigma}_i) | \forall i \in [1, V]\}$ is available (where $\boldsymbol{\xi}_i$ is a raw measurement and $\mathcal{N}(0, \boldsymbol{\Sigma}_i)$ the distribution over measurement error), one could directly minimize the distance between the predicted dis-

[1]While some noise models only consider the low dimensional synthesized measurement, this often does not capture sufficient information about the varied performance of the measurement function. For example, consider an ICP algorithm that aligns two successive laser scans, where reasoning on the information quality based solely on the state or the output of the ICP measurement can fail to capture nuances of the information in scans with varying geometric features.

tribution $\mathcal{N}(0, g(\boldsymbol{\xi}_i))$ and the true distribution $\mathcal{N}(0, \boldsymbol{\Sigma}_i)$ by minimizing a distance metric such as Kullback-Leibler divergence

$$\sum_{i=1}^{V} \mathrm{KL}(\mathcal{N}(0, g(\boldsymbol{\xi}_i)) || \mathcal{N}(0, \boldsymbol{\Sigma}_i)), \tag{4.4}$$

where the parameters of function g are varied.

However, there are two complications to this approach. First, it is often difficult to obtain the true distribution over measurement error $\mathcal{N}(0, \boldsymbol{\Sigma}_i)$ during training time. Second, the approximator function g must only produce positive definite covariance matrices, i.e., the optimization is constrained subject to $v^\mathsf{T} \boldsymbol{\Sigma}_i v > 0$ for any non-zero $v \in \mathbb{R}^n$.

To overcome these difficulties, as in White [74], we reformulate Eq. 4.4 to instead maximize the likelihood of drawing each measurement error $\boldsymbol{e}_i$ from a predicted distribution $\mathcal{N}(0, g(\boldsymbol{\xi}_i))$, and add a decomposition to the approximator function g to relax the constraint for positive definiteness in the optimization. In the following sections, we describe the details of the function approximator, the objective function, and finally the optimization techniques used to robustly predict measurement covariances without requiring difficult-to-obtain training data.

4.2.2 Deep Convolutional Neural Network Model

In order to learn the complex mapping g between the raw measurement $\boldsymbol{\xi}_i$ and the covariance matrix $\boldsymbol{\Sigma}_i$, we utilize an over-parametrized form of the approximator function. We take a similar approach to that of recent successes [36, 65] and utilize a deep convolutional neural network (CNN) with fully connected output layers, shown in Fig. 5-5, as our model.

We express the final form of the function approximator as a combination of convolution and fully connected layers. Let $f_k(\boldsymbol{\xi}; \boldsymbol{W}_1, ..., \boldsymbol{W}_k, \boldsymbol{b}_1, ..., \boldsymbol{b}_k) \mapsto \mathbb{R}^r$ be a k layer convolutional neural network, where the parameters being optimized are weight matrices $\boldsymbol{W}_{1:k}$ and bias vectors $\boldsymbol{b}_{1:k}$, as in Equation 3.11. The cascaded operations of convolution $\boldsymbol{W}_j * \boldsymbol{\xi}_i$ and non-linear activation function ϕ reduce the high dimensional

raw measurement $\boldsymbol{\xi}_i \in \mathbb{R}^m$ down to a set of low-dimensional features $f_k(\boldsymbol{\xi}_i) \in \mathbb{R}^r$, $r \ll m$.

These feature responses are analogous to the hand-coded features in [27], and as in previous work, we linearly combine them to produce a vectorized form of the covariance matrix $\boldsymbol{r}_i \in \mathbb{R}^{n \times n}$:

$$\begin{aligned} \boldsymbol{r}_i \approx g(\boldsymbol{\xi}_i) &= \boldsymbol{W}_{k+1} f_k(\boldsymbol{\xi}_i) + \boldsymbol{b}_{k+1} \\ &= f_m(f_k(\boldsymbol{\xi}_i)) \end{aligned} \tag{4.5}$$

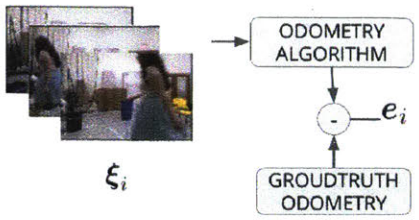
where the second line follows by using the definition of Equation 3.4.

Our method combines the optimization of feature representation (encoded in the weight matrices $\boldsymbol{W}_{1:k}$ and the bias vectors $\boldsymbol{b}_{1:k}$), as well as the weighting ($\boldsymbol{W}_{k+1}$, and $\boldsymbol{b}_{k+1}$) of the features f_k to predict the final covariance parameters $\boldsymbol{r}_i$. The optimized features are therefore specifically tailored for accurate covariance prediction and do not require hand-coding. Additionally, after the network parameters have been optimized, covariance prediction is simply the evaluation of the raw sensor data through the network, and is therefore constant time. In the following section, we discuss the objective function used to optimize the network.

4.2.3 Negative Log-Likelihood Objective Function

We would like to optimize the model presented in Equation 4.5 using supervised learning techniques. Given a training dataset $\mathcal{D} = \{\boldsymbol{\xi}_i, \mathcal{N}(0, \boldsymbol{\Sigma}_i) | \forall i \in [1, V]\}$ of pairs of raw measurement $\boldsymbol{\xi}_i$ and a distribution over measurement error $\mathcal{N}(0, \boldsymbol{\Sigma}_i)$, one could optimize for the parameters $\boldsymbol{W}_{1:k}$ and $\boldsymbol{b}_{1:k}$ of the network by directly minimizing a distance metric between the predicted and the true distributions, as in Equation 4.4.

However, it is infeasible to obtain the true distribution over sensor measurement errors. Instead, it is often much easier to obtain measurement errors $\boldsymbol{e}_i$ (drawn from $\mathcal{N}(0, \boldsymbol{\Sigma}_i)$), given a reliable sensor such as an indoor positioning system (IPS) that can obtain the same measurement $\boldsymbol{z}_i^*$ with higher precision and accuracy. For example, with a highly precise and accurate IPS, one could calculate the odometry between



$$\mathcal{D} = \{\boldsymbol{\xi}_i, \boldsymbol{e}_i | \forall i \in [1, V]\}$$

Figure 4-1: To generate the training dataset, raw sensor measurements are fed to an odometry algorithm. The error label is the difference between the groundtruth odometry and the estimated odometry.

two timesteps, and consider this measurement to be the groundtruth measurement. Therefore, one can more conveniently obtain a training dataset $\mathcal{D} = \{\boldsymbol{\xi}_i, \boldsymbol{e}_i | \forall i \in [1, V]\}$ where $\boldsymbol{e}_i$ is the error made by the sensor, i.e., $\boldsymbol{e}_i = |\boldsymbol{z}_i^* - \boldsymbol{z}_i|$. Figure 4-1 illustrates an example pipeline for generating the training dataset.

Given the training dataset $\mathcal{D} = \{\boldsymbol{\xi}_i, \boldsymbol{e}_i\}$ of raw measurements $\boldsymbol{\xi}_i$ and measurement errors $\boldsymbol{e}_i$, we can instead maximize the likelihood of drawing the measurement errors from the distribution over the errors

$$\underset{\boldsymbol{\Sigma}_{1:V}}{\operatorname{argmax}} \sum_{i=1}^{V} p\left(\boldsymbol{e}_i | \boldsymbol{\Sigma}_i\right), \tag{4.6}$$

or minimize the negative log-likelihood

$$J = \underset{\boldsymbol{\Sigma}_{1:V}}{\operatorname{argmin}} \sum_{i=1}^{V} -\log\Big(p\left(\boldsymbol{e}_i | \boldsymbol{\Sigma}_i\right)\Big) \tag{4.7}$$

$$= \underset{\boldsymbol{\Sigma}_{1:V}}{\operatorname{argmin}} \sum_{i=1}^{V} \log|\boldsymbol{\Sigma}_i| + \boldsymbol{e}_i^{\mathsf{T}} \boldsymbol{\Sigma}_i^{-1} \boldsymbol{e}_i. \tag{4.8}$$

$$\approx \underset{\boldsymbol{W}_{1:k}, \boldsymbol{b}_{1:k}}{\operatorname{argmin}} \sum_{i=1}^{V} \log|g(\boldsymbol{\xi}_i)| + \boldsymbol{e}_i^{\mathsf{T}} g(\boldsymbol{\xi}_i)^{-1} \boldsymbol{e}_i \tag{4.9}$$

where $\boldsymbol{W}_{1:k+1}, \boldsymbol{b}_{1:k+1}$ are the parameters of function g, i.e., the parameters of the neural network being optimized.

4.2.4 Parameterization for Positive Definiteness

The optimization in Eq. 4.9 is subject to all predictions $g(\boldsymbol{\xi}_i)$ being positive definite matrices. We would like to remove this constraint so that the function can be optimized using standard SGD techniques. To do this, we reformulate g to predict a decomposition of the covariance matrix $\boldsymbol{\Sigma}_i$, i.e., the free parameters $\boldsymbol{\alpha}_i \in \mathbb{R}^q$ where $q = \frac{n(n+1)}{2}$, instead of its vectorized form $\boldsymbol{r}_i \in \mathbb{R}^{n \times n}$. We then add a known function g_d that re-constructs the covariance matrix, i.e., splitting Eq. 4.3 into $\boldsymbol{\Sigma}_i \approx g_d(g_h(\boldsymbol{\xi}_i))$, where g_h is the new predictor function for the free parameters $\boldsymbol{\alpha}_i$:

$$\boldsymbol{\alpha}_i \approx g_h(\boldsymbol{\xi}_i) = \phi(\boldsymbol{W}_{k+1} \times f_k(\boldsymbol{\xi}_i) + \boldsymbol{b}_{k+1}). \tag{4.10}$$

We choose the LDL decomposition as in [27],

$$\boldsymbol{\Sigma}_i \approx g_d(\boldsymbol{\alpha}_i) = L(\boldsymbol{l}_i) D(\boldsymbol{d}_i) L(\boldsymbol{l}_i)^\mathsf{T} \tag{4.11}$$

where the free parameters $\boldsymbol{\alpha}_i = \left[\boldsymbol{d}_i, \boldsymbol{l}_i\right]^\mathsf{T}$ consist of a sub-vector $\boldsymbol{d}_i \in \mathbb{R}^n$ for reconstructing the diagonal matrix, and a sub-vector $\boldsymbol{l}_i \in \mathbb{R}^{(n^2-n)/2}$ for the lower unitriangular matrix. This decomposition exists and is unique for all positive definite matrices as long as the diagonal elements of $D(\boldsymbol{d}_i)$ are constrained to be positive. A simple way to enforce this constraint is to add an element-wise exponential function to the diagonal vector, i.e., update the diagonal matrix to be $D(\exp(\boldsymbol{d}_i))$. Explicitly, we have:

$$b = \frac{n^2 - n}{2} \tag{4.12}$$

$$\alpha_i = \begin{bmatrix} d_{i_1} \\ \vdots \\ d_{i_n} \\ l_{i_1} \\ \vdots \\ l_{i_b} \end{bmatrix} \tag{4.13}$$

$$D(\exp(d_i)) = \begin{bmatrix} \exp(d_{i_1}) & 0 & 0 & 0 \\ 0 & \exp(d_{i_2}) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \exp(d_{i_n}) \end{bmatrix} \tag{4.14}$$

$$L(l_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{i_1} & 1 & 0 & 0 \\ \vdots & \ddots & \ddots & 0 \\ l_b & \dots & l_{i_{n-1}} & 1 \end{bmatrix} \tag{4.15}$$

While any decomposition that does not impose difficult constraints on the free parameters $\boldsymbol{\alpha}_i$ would suffice, the LDL decomposition is particularly attractive due to its numeric stability in computing the log-determinant:

$$\begin{aligned} \log|\boldsymbol{\Sigma}_i| &= \log|L(\boldsymbol{l}_i)D(\exp(\boldsymbol{d}_i))L(\boldsymbol{l}_i)^\mathsf{T}| \\ &= \log\big(|L(\boldsymbol{l}_i)\big)||D(\exp(\boldsymbol{d}_i))||L(\boldsymbol{l}_i)^\mathsf{T}| \\ &= \log\left((1)(\prod_i^n \exp(d_i))(1)\right) \\ &= ||\boldsymbol{d}_i||_1, \end{aligned} \tag{4.16}$$

where $||\boldsymbol{d}_i||_1$ denotes the summation of the elements of the vector v. By predicting the parameters of the a positive definite matrix, the search is now explicitly limited to the hyperspace of positive definite matrices.

4.2.5 Optimization

The final optimization problem is then

$$\begin{aligned}\operatorname{argmin} \boldsymbol{W}_{1:k+1}, \boldsymbol{b}_{1:k+1} & \sum_{i=1}^{V} \operatorname{sum}(\boldsymbol{d}_i) \\ & + \boldsymbol{e}_i^{\mathsf{T}} (L(\boldsymbol{l}_i) D(\exp(\boldsymbol{d}_i)) L(\boldsymbol{l}_i)^{\mathsf{T}})^{-1} \boldsymbol{e}_i,\end{aligned} \tag{4.17}$$

where $g_h(\boldsymbol{\xi}_i) \approx \boldsymbol{\alpha}_i = \left[\boldsymbol{l}_i, \boldsymbol{d}_i\right]^{\mathsf{T}}$.

To optimize the objective function given in Equation 4.9 we must have a way to select the best weight and bias parameters. We use non-linear activation functions, it is therefore necessary to perform this optimization iteratively; the weights of the model are updated using a gradient descent method, where each iterative update to the weight w_{ij}^{t+1} is based on the value at the previous iteration w_{ij}^{t} and the value of the gradient evaluated at the last estimate (Equation 3.13). In this work, we use the Nesterov Accelerated Gradient (NAG) variant of the method [49]. To prevent overfitting to the data, we utilize dropouts as a form of regularization. These implementation details are briefly discussed in the following sections.

Nerostov Accelerated Gradient

As outlined in Sutskever et al. [70], w_{ij}^{t+1} is calculated by adding a velocity term to the previous estimate:

$$w_{ij}^{t+1} = w_{ij}^{t} + v_{ij}^{t+1} \tag{4.18}$$

where v_{ij}^{t+1} is calculated as

$$v_{ij}^{t+1} = \Gamma v_{ij}^{t} - \gamma \frac{\mathrm{d}J}{\mathrm{d}w_{ij}}, \tag{4.19}$$

The parameters γ and Γ are often called the *learning rate* and *momentum rate*, respectively. The update to the bias vectors are similarly calculated. Due to the volume of data we use in this work, we use Stochastic Gradient Descent (SGD) with mini-batches, where the gradient is calculated from a randomly selected subset of the training data at each iteration, for computational tractability. In order to evaluate

Equation 4.18 and update each of the weights, we use backpropagation[2] to calculate $\frac{\mathrm{d}J}{\mathrm{d}w_{ij}}$ for each weight in the model.

Regularization

We would like to regularize the output of the network to encourage smoothness in the solution and prevent over-fitting. Hu et al. rely on a regularization of α (the output terms of the network). We instead use dropouts to regularize the model predictions. Proposed in Srivastava et al. [67], regularization via dropout involves stochastically removing the output from certain nodes and has shown promise in improving network performance on tasks such as image classification and speech classification [25], [12]. In practice, dropout is typically used for fully connected perceptron layers; this is the formulation we present here. Specifically, as shown by Srivastava et al., node i in the perceptron layer l has a probability of dropping out characterized by a Bernoulli distribution parameterized by p. Let the sampled dropout value of o_{l_i} be denoted r_{l_i}. We then have

$$r_{l_i} \sim \mathrm{Bern}(p), \tag{4.20}$$

where $r_{l_i} \in 0, 1$. The vector of dropout indicators is then multiplied against the output of the layer, i.e.,

$$\hat{\boldsymbol{o}}_l = \boldsymbol{r}_l * \boldsymbol{o}_l. \tag{4.21}$$

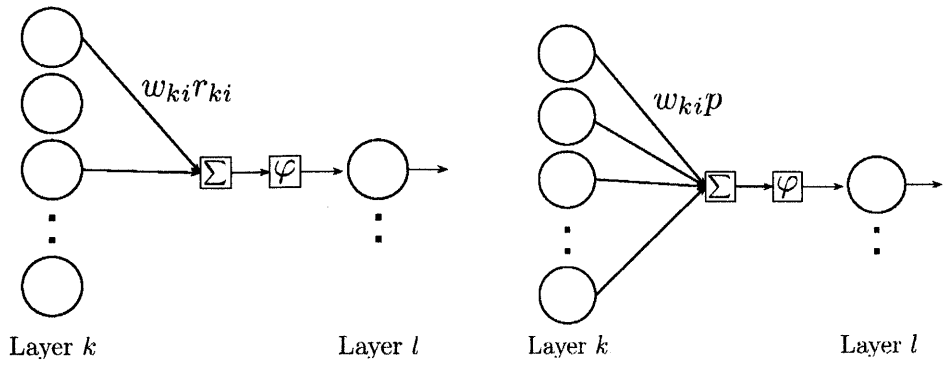
The new output of the layer, $\hat{\boldsymbol{o}}_l$ "drops" the original output of randomly selected nodes.

Each optimization iteration in the training stage, a subset of nodes are randomly dropped:

$$\boldsymbol{o}_{l,train} = \varphi\Big(\boldsymbol{W}_{lk}\hat{\boldsymbol{o}}_k + \boldsymbol{b}_{lk}\Big). \tag{4.22}$$

Back-propagation is done only over the nodes that were not dropped in the forward pass.

[2]See subsection 3.2.2



(a) Forward propagation phase (b) Query time

Figure 4-2: Effects of dropouts on weights during train time vs. query time. During the forward propagation phase (a), weights are randomly set to zero, as defined by r_{ki} (Equation 4.20). After the model has been optimized (b), each edge weight is then multiplied by the hyper-parameter of the Bernoulli distribution p used for dropouts at the training stage.

In contrast, at test time, each optimized edge weight $\boldsymbol{W}^*_{lk}$ is multiplied by p:

$$\boldsymbol{o}_{l,test} = \varphi\Big(\hat{\boldsymbol{W}}^*_{lk}\boldsymbol{o}_k + \boldsymbol{b}_{lk}\Big), \tag{4.23}$$

with the modified weight matrix defined as

$$\hat{\boldsymbol{W}}_{lk} = p * \boldsymbol{W}_{lk}, \tag{4.24}$$

where every element of W is multiplied by p.

4.2.6 System Overview

DICE is a two stage algorithm, which we distinguish as the *training* stage and the *query* or *test* stage (see Figure 4-3). In the training stage, pairs of raw sensor data and measurement error labels are used to optimize the weights of the hyper-parametric function approximator with a log-likelihood loss function. In the second stage, raw sensor measurements are passed through the function approximator with optimized

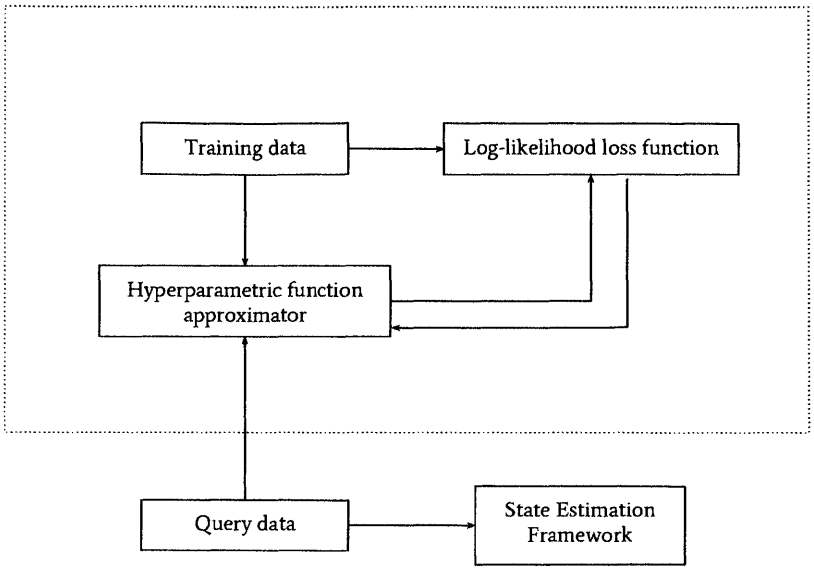


Figure 4-3: DICE system diagram. The processes in the dotted lines are used in the offline training stage.

weights to predict the parameters of the measurement covariance. The covariance estimates can then be used in a state estimation framework.

In the following section, we evaluate the accuracy of the covariance prediction function learned from solving the optimization problem.

Chapter 5

Experiments

To evaluate the performance of using deep neural networks to estimate the measurement noise models for complex sensors, we tested the approach in both simulation and real-world environments. For the simulation environment, we report the Kullback-Leibler divergence between the predicted covariances and the groundtruth covariance. For the real-world environment, we train the network on a frame-to-frame visual odometry task, and report log-likelihood results. We then use the estimated covariances in a pose-graph estimation framework to illustrate the benefit of accurate covariance estimation in a real-world task.

5.1 Simulation Results

To validate our approach, we first picked a virtual environment where we could simulate a sensor with known measurement covariances. Ground-truth covariances are normally difficult to obtain, but using a simulation environment circumvented this difficulty.

5.1.1 Simulation Setup

We chose to use the same simulation environment as in Vega-Brown et al. [73], where the simulated position sensor's performance is correlated with the brightness of the

location $\boldsymbol{x} \in \mathbb{R}^2$ within a known 2D map $\boldsymbol{m} \in \mathbb{R}^{k \times k}$, i.e.,

$$\boldsymbol{\Sigma} = f(\boldsymbol{x}, \boldsymbol{m}). \tag{5.1}$$

We designed the sensor noise model f to include exponential and cosine components, introducing complex non-linear noise characteristics. We then randomly sampled 1000 locations $\boldsymbol{x}_i$ within a known map $\boldsymbol{m}$ and for each sample, computed the measurement covariance $\boldsymbol{\Sigma}_i$ from which to draw an error label $\boldsymbol{e}_i$, i.e.,

$$\boldsymbol{e}_i \sim \mathcal{N}(0, f(\boldsymbol{x}_i, \boldsymbol{m})). \tag{5.2}$$

We chose the local region of the map $\boldsymbol{m}$ around the robot position $\boldsymbol{x}_i$ to be the raw measurement $\boldsymbol{\xi}_i$, i.e.

$$\boldsymbol{\xi}_i = h(\boldsymbol{x}_i, \boldsymbol{m}), \tag{5.3}$$

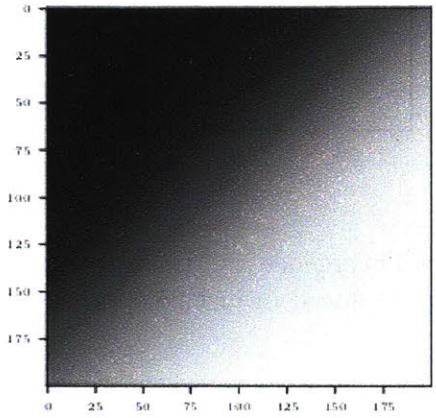
and obtained the training set $\mathcal{D} = \{\boldsymbol{\Sigma}_i, \boldsymbol{e}_i, \boldsymbol{\xi}_i | \forall i \in [1, V]\}$. The local region is simply a 10×10 patch of pixels roughly centered at the measurement location[1]. Example measurements are shown in Figure 5-1.

5.1.2 Simulation Results

We optimized the objective function g in Equation 4.17 using SGD. Due to the simplicity of this problem, we were able to reduce the complexity of the function, removing convolution and reducing the network size. The network was implemented using the open-source neural network library, Lasagne [14]. After optimizing for 25 epochs, we obtained predicted covariances $g(\boldsymbol{\xi}_i)$ for each raw measurement $\boldsymbol{\xi}_i$. With access to ground-truth covariance labels $\boldsymbol{\Sigma}_i$, we evaluated the Kullback-Leibler divergence (KLD) in Equation 4.4 at each epoch.

As shown in Fig. 5-2, minimizing the negative log-likelihood quickly reduced KLD, an evidence of the alternative optimization in Equation 4.6 reducing the direct

[1]Due to discretization, we arbitrarily choose the pixel location (6,6) in the measurement to correspond to the measurement location.



(a) Simulation environment map.

(b)

(c)

Figure 5-1: Example raw measurements for the DarkRoom environment. Each measurement is a local pixel patch from a larger map, which is visualized in subfigure (a). Subfigures (b) and (c) are examples of 10 by 10 measurements at $(118.6, 58.2)$ and $(40.5, 88.3)$ respectively.

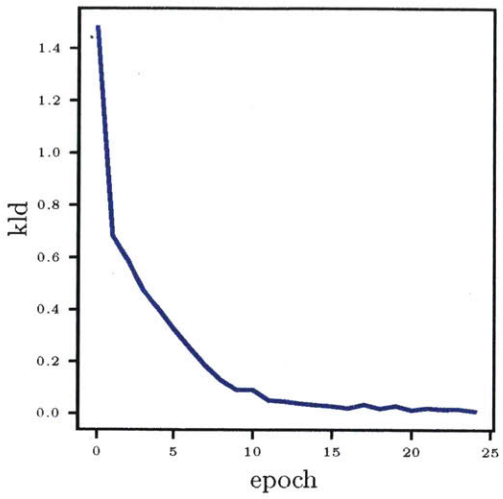


Figure 5-2: Kullback-Leibler Divergence vs. training epoch for the simulation environment. The KLD is quickly reduced in just 25 epochs, supporting the convergence properties seen visually in Figure 5-3.

distance metric. We also compared a few representative covariance predictions against the ground-truth for qualitative evaluation at various epochs. These are visualized in Figure 5-3.

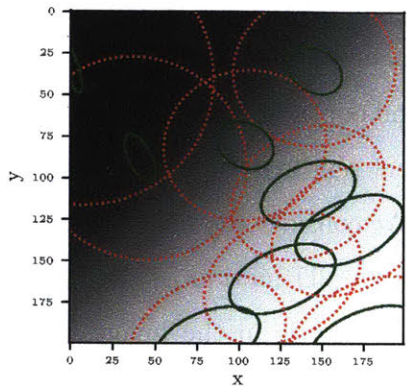
5.2 Experiment Results

In this section, we benchmark our method against several other covariance estimation techniques for the task of estimating covariances for a visual odometry algorithm.

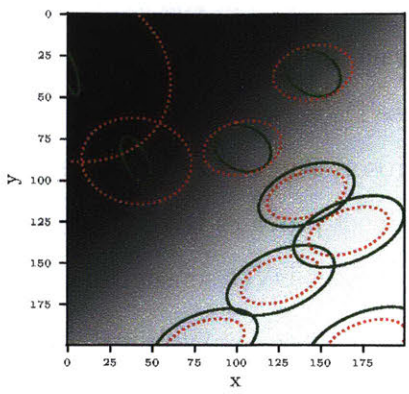
5.2.1 Experiment Setup

Direct Visual Odometry (DVO) Overview

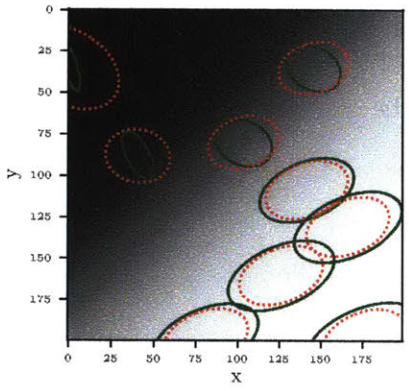
We evaluated the performance of DICE on predicting the measurement covariances of the output of a 2D visual odometry (VO) algorithm based on DVO [31]. The algorithm densely aligns each new RGB image to the previous image by projecting the image into the world using an associated depth measurement, then solving for the best back-projection into the previous image that minimizes the photometric error



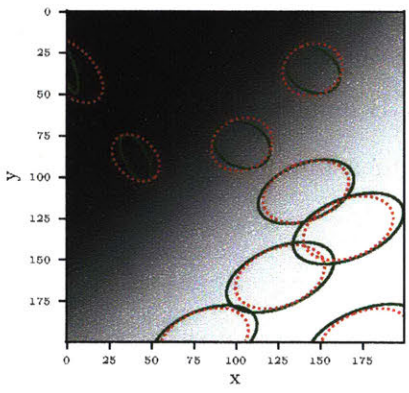
(a) Epoch 0



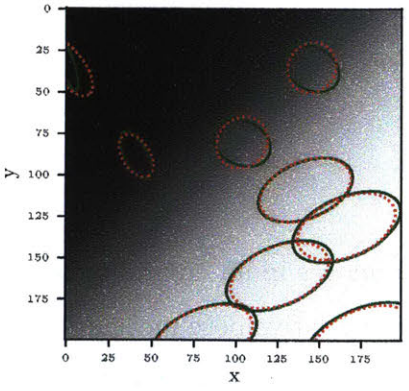
(b) Epoch 1



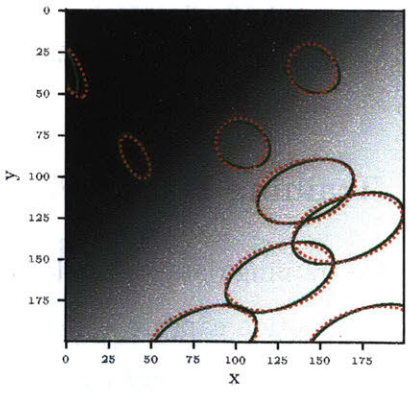
(c) Epoch 5



(d) Epoch 7



(e) Epoch 9



(f) Epoch 11

Figure 5-3: Visualization of convergence of covariance estimates. Predicted covariance ellipses (red) and ground-truth covariance ellipses (green) are shown for a few samples in the simulated map.

between the two. The difference between our 2D VO algorithm and standard DVO is that back-projection is limited to 2D motions, and no motion prior is used[2].

To use DVO, we must have access to RGB and depth images. Therefore, we chose the Microsoft Kinect [78] as the raw measurement sensor to pair with the algorithm, as the Kinect provides RGB-D measurements. We consider the sensor-algorithm pair to be a Kinect-DVO sensor that measures relative 2D motion. Based on the fact that the odometry is computed by minimizing a metric in the image-space (while the depth information is only used for projections) and the similarity of consecutive RGB images under moderate motion, we assumed that most predictive features of uncertainty are in the latest RGB image. We therefore chose only the latest RGB image as the input to our predictor function g, although the VO algorithm required both RGB images and a depth image to solve the alignment.

The measurement process is then modeled as

$$\boldsymbol{x}_i^{-1} \oplus \boldsymbol{x}_{i-1} = h_{DVO}(\boldsymbol{\xi}_i) + \boldsymbol{w}_i \tag{5.4}$$

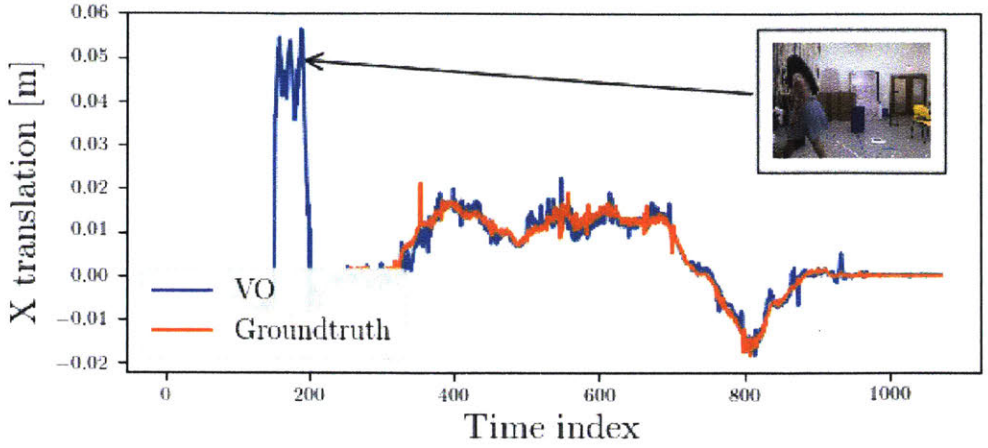
$$\boldsymbol{w}_i \sim \mathcal{N}(0, \boldsymbol{\Sigma}_i) \tag{5.5}$$

where $\oplus$ denotes pose composition in SE(3). In terms of previous filtering frameworks, the measurement is the relative transform between the two state: $\boldsymbol{z}_i = \boldsymbol{x}_i^{-1} \oplus \boldsymbol{x}_{i-1}$.

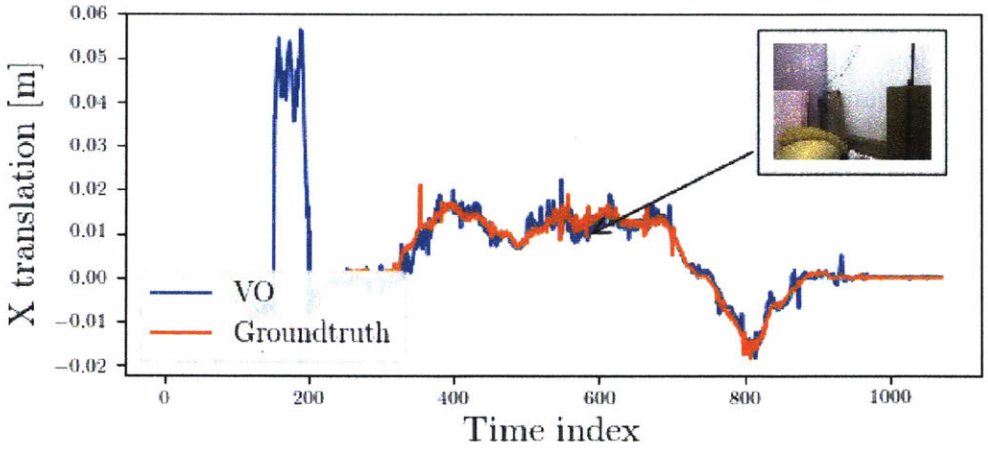
5.2.2 Training Data Collection

To generate training data, we collected RGB-D images in an environment equipped with a Vicon motion capture system. We obtained relative pose measurements using the Kinect-DVO sensor, and computed the error in the measurements as deviation from the Vicon measurements. We collected approximately 42,500 pairs of images and error labels as training data. In this experiment, we specifically explored two

[2]We observe here that our method requires that the measurement model to be frame-to-frame, or a function of only the two most recent measurements. Explicitly considering more than one prior measurement into the prediction is left to future work.



(a) Dynamic training example



(b) Textureless training example

Figure 5-4: Visualization of selected training data examples. In these plots, the groundtruth odometry is shown in orange, and the measured odometry shown in blue. Dynamic objects can be seen to induce more error in the odometry estimates.

common failure modes of VO algorithms: low texture scenes and dynamic scenes. The environment was set up to include varying degrees of texture. In some parts of the dataset, a person periodically walked in and out of the sensor frame during data collection. Examples of training data are shown in Figure 5-4. The objects in the environments were also moved around to prevent over-fitting to the environment. The sensor was constrained to move in a 2D plane.

5.2.3 Network Structure and Training

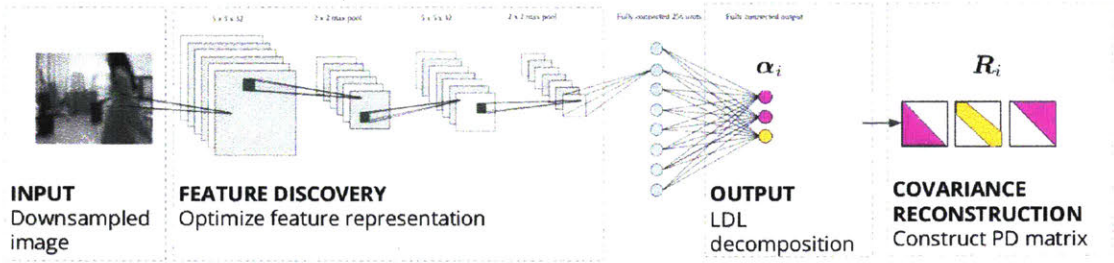


Figure 5-5: Neural network architecture used to approximate the covariance prediction function. The input to the network is the raw measurement, e.g., a (possibly) down-scaled camera image, and the output is the vector of the free parameters of the covariance. The narrowing architecture in the beginning reduces the raw measurement into low-dimensional features, and the following fully-connected layer optimally combines the features for the covariance prediction task.

While representing the covariance predictor as a CNN removes the need to carefully specify a parametric form of the covariance approximation function, high-level design choices (width, depth, etc) are required for the network.

We used an eight layer deep CNN and the input RGB images were down-sampled to 48x64 pixel grey-scale images. Each of the two convolutional layers consisted of 32 kernels of size 5x5, followed by a 2x2 max pooling layer. Before the output, there was one fully connected layer of 256 units, and a dropout layer with a dropout rate of 50% to prevent over-fitting. As with the simulation experiments, the neural network was implemented using Lasagne. The network, visualized in Fig. 5-5, was trained using an Nvidia GeForce GTX 1080 for 6000 epochs, with a learning rate of 0.0001, a Nesterov momentum of 0.9, and a mini-batch size of 500. After experimenting with

different nonlinear activation functions, we found that a leaky rectify activation [39] provided the most stable optimization process.

5.2.4 Test Set Selection

We chose four test sets to benchmark DICE. The first three, *Dynamic*, *White-wall* and *Unseen-Dynamic*, were drawn from the environments in the training data. The third is an open-source dataset. For reporting the likelihood metric, we removed data points where Vicon measurements were measurably wrong (approximately 2% of the measurements).

Dynamic Objects Dataset

In this dataset, the sensor moved in a box pattern around an environment present in the training dataset. Early in the trajectory, as the Kinect was moving forward, a person present in the training dataset walked across the camera frame one time. The sensor completed the rest of the trajectory without any dynamic disturbances. This test set is intended to test covariance estimation of odometry estimates for dynamic scenes.

White Wall Dataset

The *White-wall* dataset was also collected in an environment present in the training dataset. In this dataset, the Kinect moved in a loop with two separate low texture scenes. The White-wall dataset is designed to test covariance prediction performance for texture-less scenes.

Unseen-Dynamic

The *Unseen-Dynamic* dataset was collected in an environment present in the training dataset. The Kinect moved in a loop; in two instances, a person walked through the sensor frame. Unlike the Dynamic objects test set, the person who walked into the

field of view was not present in the training dataset. The Dynamic-Unseen test set is designed to test covariance estimation on partially novel scenarios.

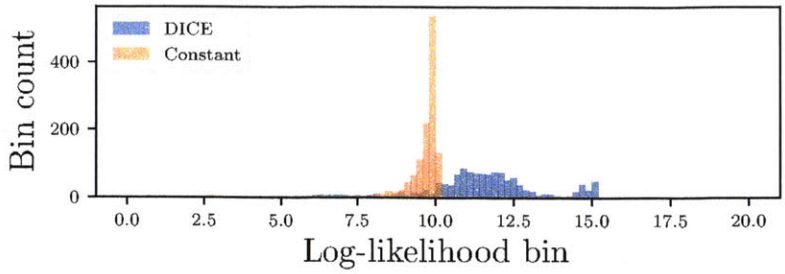
TUM

We denote the *freiburg3_walking_static* dataset (from the *TUM* RGB-D SLAM benchmarks [69]) as the TUM dataset. In this dataset, the Kinect was static and two people walked in and out of frame. We choose this dataset in particular due to 3DOF assumption made in the odometry optimization; because the Kinect in this dataset is static, this does not violate the 3DOF assumption. This dataset was chosen to test the generalization potential of different covariance estimation techniques.

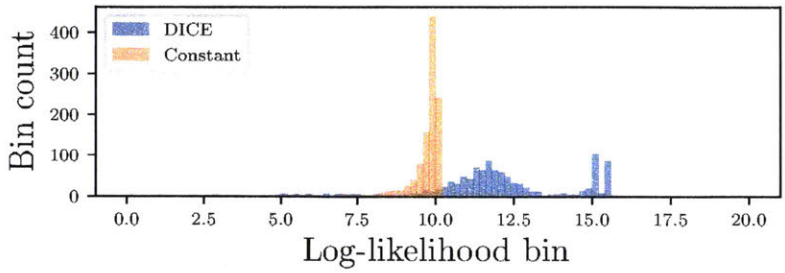
5.2.5 Measurement Likelihood Performance

In real environments, we do not have access to the true distribution over measurement error to compare directly against the predicted distributions as done in Section 5.1. Instead, we can compare the likelihood of drawing true measurement errors from predicted distributions as in Equation. 4.6.

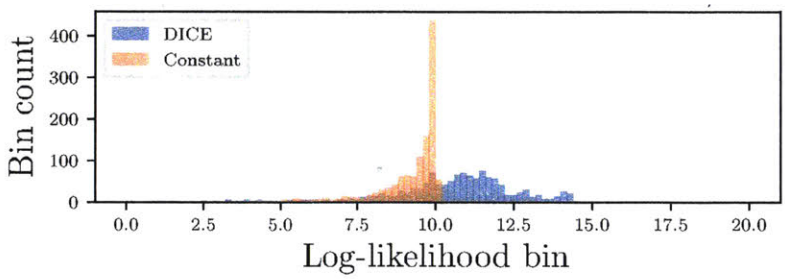
We benchmarked the covariance prediction performance of DICE against a constant covariance model, DVO's internal prediction based on the Cramer-Rao bound [58], and CELLO. To determine an appropriate constant covariance, we calculated a single covariance over all of the training data. The Cramer-Rao covariances were obtained by inverting the Fisher information matrices of the dense photometric alignment and multiplying by an empirically-determined factor to compensate for the bounds being lower-bounds and largely conservative. We used the empirical factor provided in the open-source implementation of DVO, noting that this parameter can also be tuned to perform arbitrarily well on any given dataset. The CELLO covariances were obtained using 10 image-space features proposed in [72] (e.g. dynamic range, pixel entropy, image gradients, etc) and the entire training dataset for DICE was used as potential neighbors in the kernel estimator. A few representative constant and DICE measurement covariances are shown in Fig. 5-8.



(a) Dynamic



(b) Unseen-Dynamic

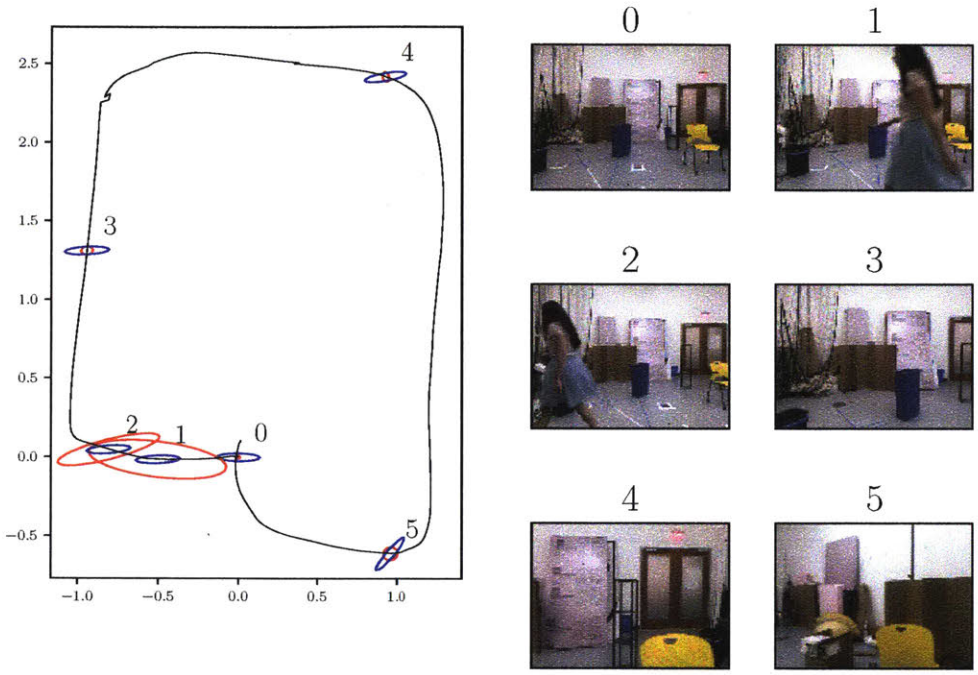


(c) White-wall

Figure 5-6: Log-likelihood histograms comparing DICE and constant covariance prediction performance.

Table 5.1: Mean log-likelihood results are shown for each prediction method and dataset pair. Values marked with † were generated after the prediction method was further trained with the augmented training dataset. On average, DICE predicted covariances better model the errors over all four test sets.

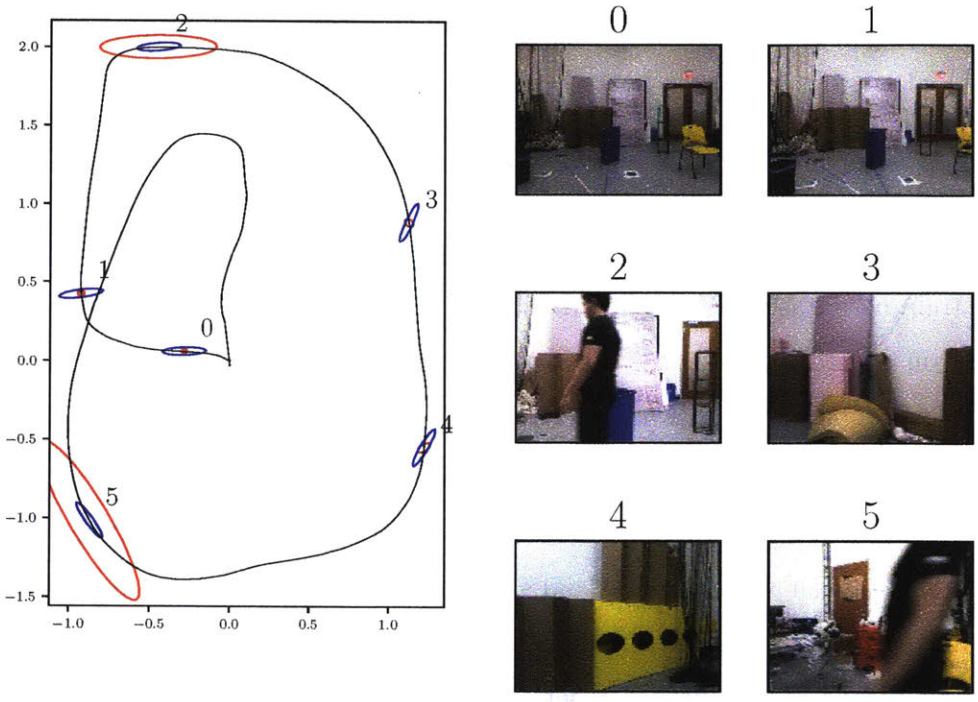
	Dynamic	*White-wall*	*Unseen-Dynamic*	*TUM*
Constant	9.20	8.84	9.27	8.62
Cramer-Rao	-26.72	-2.92	-29.06	4.95
CELLO	10.61	9.52	10.36	9.15†
DICE	**11.80**	**10.38**	**11.13**	**9.63**†



(a) Constant and predicted covariances

(b) RGB images

Figure 5-7: Constant (blue) and DICE (red) measurement covariances are plotted for the *Dynamic* dataset. The thumbnail images show the RGB frames at the correspondingly labeled points on the ground-truth trajectories (black). The constant covariance, which is empirically fit to the training data and non-spherical, underestimates the error in non-static scenes (i.e., 1 and 2 in *Dynamic* and overestimates in static scenes. DICE covariances vary largely both in magnitude and shape (larger and elongated in non-static scenes, while small and spherical in static scenes), more accurately representing the distribution of measurement error in different scenes. The major and minor axes of the covariance were enlarged by a factor of 20 for visualization purposes.



(a) Constant and predicted covariances

(b) RGB images

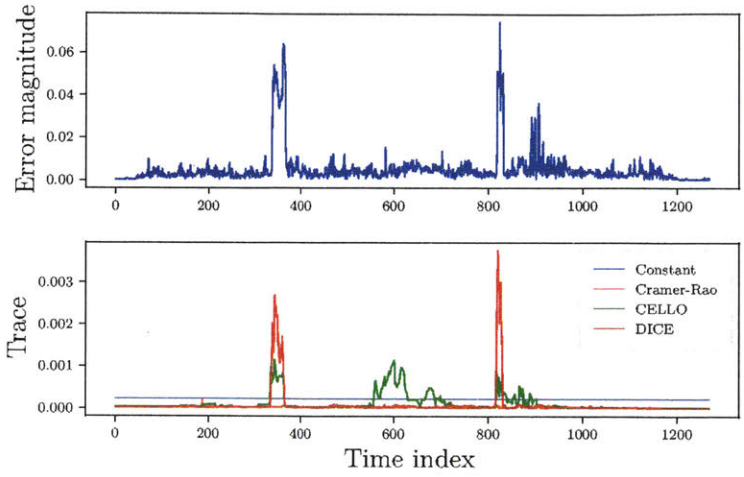
Figure 5-8: Constant (blue) and DICE (red) measurement covariances are plotted for the *Unseen-Dynamic* (right) dataset. In this test case, the constant covariance underestimates the error in scenes 2 and 5, which correspond to images where a previously unseen person walks through the sensor frame. The major and minor axes of the covariance were enlarged by a factor of 20 for visualization purposes.

Table 5.1 summarizes the performance of the four methods on each test set described in Section 5.2.4. We observed that on average, the covariances estimated by DICE better explained the observations than those of the other approaches. Fig. 5-9 illustrates the predictive performance of DICE on *Dynamic* test set; regions of the trajectory characterized by larger measurement error magnitudes were paired with larger covariance estimates by DICE. In comparison, the constant covariance method was unable to adapt, the Cramer-Rao approach was still a severe underestimate, and CELLO adapted but underestimated the magnitude of the covariance.

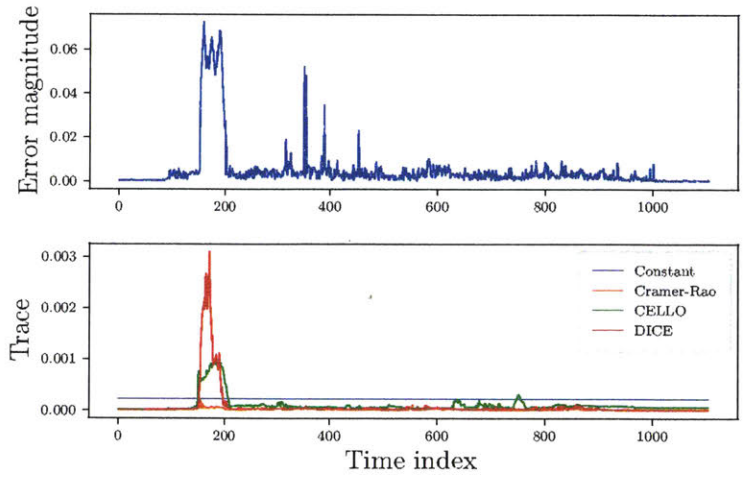
To test how well our method generalizes to new scenarios and environments, we considered the *Unseen-Dynamic* dataset. Although this dataset was set in the same environment as was used to train DICE, the person in this test set was not present in the training dataset. We observed that DICE predicted more likely measurement models, on average, than the other covariance prediction methods, indicating that the low-level representation was general to some degree.

To test the adaptability of our approach, we considered the *TUM* dataset, which has both an entirely new environment and new people. We augmented the original training dataset with the first 400 samples of the *TUM* dataset, and tuned the pre-trained network with the augmented dataset (for comparison purposes, we also augment the CELLO training dataset). The average log-likelihood results over the entire *TUM* dataset are reported in Table 5.1. Fig. 5-11 illustrates that given examples of a new environment, DICE can be re-optimized to predict better measurement models than constant, Cramer-Rao, and CELLO models, better distinguishing the measurement error in static and dynamic scenes. This example further illustrates a strength of data-driven feature discovery, providing evidence that the predictions can be further improved in new environments by taking pre-trained models and resuming optimization on new data.

We also considered the distribution of the log-likelihood performance over the test sets, and specifically compared constant and DICE predicted covariances to highlight the benefits of our heteroscedastic covariance approach. As visualized in Figure 5-6, we see that constant covariances have an implicit limit to their ability to explain



(a) Dynamic



(b) Dynamic-Unseen

Figure 5-9: The trace of four prediction methods on the dynamic objects test datasets with the label error magnitude for comparison.

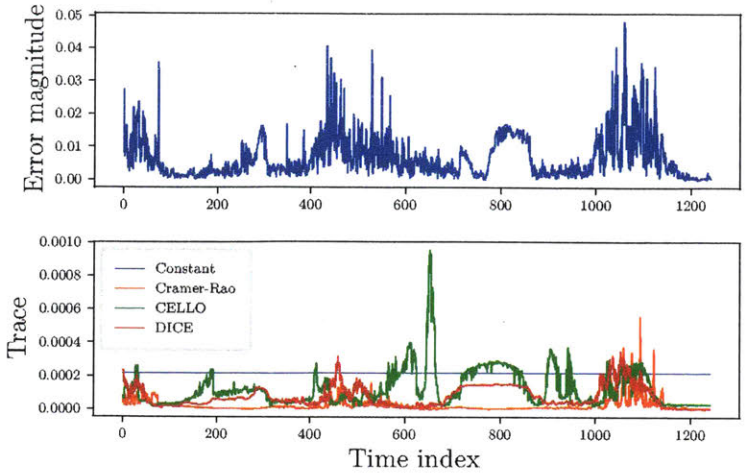


Figure 5-10: The trace of four prediction methods on the texture-less test dataset with the label error magnitude for comparison.

observations. In contrast, our predictive method, which considers the covariance to be a function of the measurement itself has not only a larger mean log-likelihood, but also a smoother distribution.

5.2.6 Trajectory Estimation Performance

To demonstrate the utility of accurately predicting measurement covariances, we additionally benchmarked DICE in a standard pose-graph optimization framework [38]

$$\operatorname{argmin} \boldsymbol{x}_{0:N} \sum_{i}^{N} \|\boldsymbol{z}_i \oplus \boldsymbol{x}_i^{-1} \oplus \boldsymbol{x}_{i-1}\|^2_{\boldsymbol{\Sigma}_i} + \|s^* \oplus \boldsymbol{x}_0^{-1} \oplus \boldsymbol{x}_N\|^2_{\boldsymbol{S}_i}, \tag{5.6}$$

Our DVO-based VO measurements $\boldsymbol{z}_i = \boldsymbol{x}_i^{i-1}$ were incorporated as measurement factors between states (i.e., odometry measurements)[3]. The ground-truth loop-closure s^* between the first and the last pose were used to optimize 2D poses $\boldsymbol{x}_i \in \mathrm{SE}(2)$. The source of measurement covariances $\boldsymbol{\Sigma}_i$ was varied to the different methods, and the loop-closure covariance $\boldsymbol{S}_i$ was chosen to be orders of magnitude smaller.

[3]We observe that this differs from the smoothing optimization discussed in 2.1.2 only in that the conditional probabilities are of measurements between poses, rather than of a the measurement obtained by a single pose.

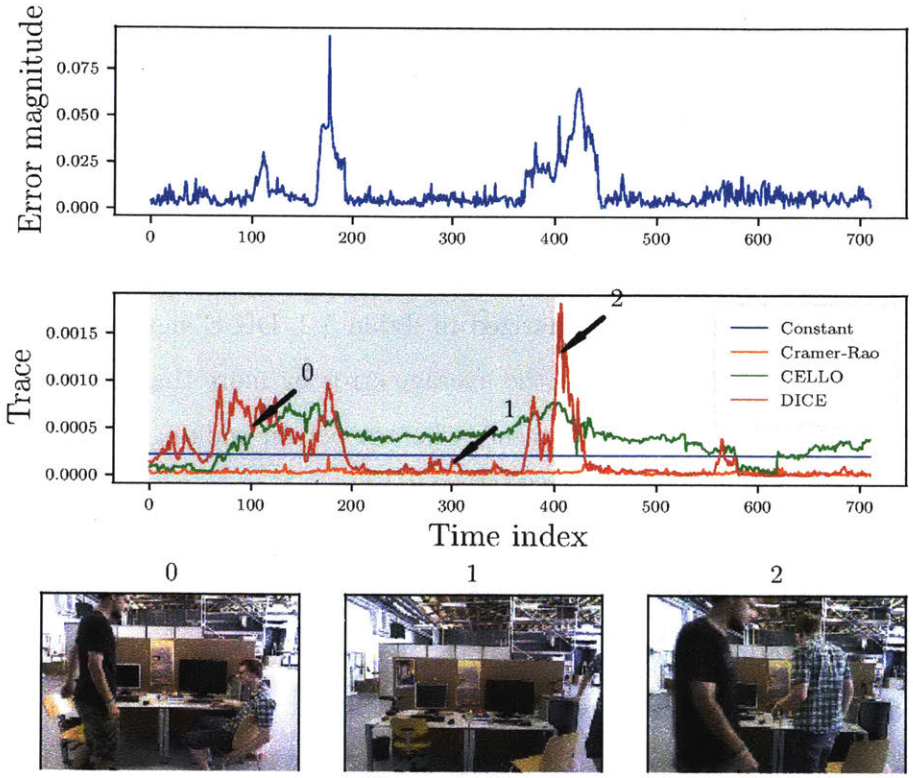


Figure 5-11: The trace of the predicted measurement covariances of all four methods on the the TUM dataset are shown, with representative images for small (1) and larger (0 and 2) magnitude trace predictions. The grey shaded region indicates the samples that were used to augment the existing training dataset, while the unshaded portions were reserved for testing. DICE outperforms the other three methods, predicting larger covariances in regions of higher error.

Table 5.2: Mean and standard error for the absolute positional error (meters) in the loop-closed trajectory are shown. DICE outperformed constant and Cramer-Rao approaches on every dataset, and for the *Dynamic* dataset where the error in the raw odometry was the greatest, the performance gain using DICE was also the greatest. In this dataset, the mean positional error in DICE was less than half of that of constant. Comparing DICE to CELLO, in datasets where CELLO's features were inadequate (dynamic scenes), DICE outperformed CELLO. For the dataset where the CELLO features were representative, DICE was tied with CELLO.

	Dynamic	*White-wall*	*Unseen-Dynamic*
Constant	0.57 ± 0.015	0.74 ± 0.013	0.70 ± 0.012
Cramer-Rao	0.59 ± 0.015	0.77 ± 0.011	0.67 ± 0.011
CELLO	0.37 ± 0.014	$\mathbf{0.50 \pm 0.0093}$	0.45 ± 0.010
DiCE	$\mathbf{0.25 \pm 0.011}$	$\mathbf{0.50 \pm 0.010}$	$\mathbf{0.39 \pm 0.010}$

We compared the mean and the standard error of the absolute positional error in the loop-closed trajectory. Summarized in Table 5.2, DICE significantly outperformed the other methods, reducing the average error by more than a factor of two in the *Dynamic* dataset. In this dataset, where a person walking across the camera corrupted the raw odometry, DICE distrusted the dynamic region and allowed the rest of the trajectory to be recovered more accurately (shown in Figure ??). In comparison, CELLO distrusted the region noticeably less than DICE, while constant and Cramer-Rao methods did not distrust the dynamic region, resulting in the error being distributed across the entire trajectory.

Similar results were obtained for the *Unseen-Dynamic* test set, where a previously unobserved person came into the view two times. Unlike the CELLO predicted trajectory, is better to recover the initial portion of the trajectory; this suggests that DICE is able to more precisely identify the boundary between scenes with high and low sensor-environment measurement error. The results for this test set are visualized in Figure 5-9.

Additionally, in the *White-wall* dataset (visualized in Figure 5-14), DICE performed significantly better than constant and Cramer-Rao methods, but was on par with CELLO. We observed that for failure modes that are well explained by the hand-coded features of CELLO, i.e., white walls form a tight cluster in the feature space defined using image-space statistics, CELLO could perform as well as DICE.

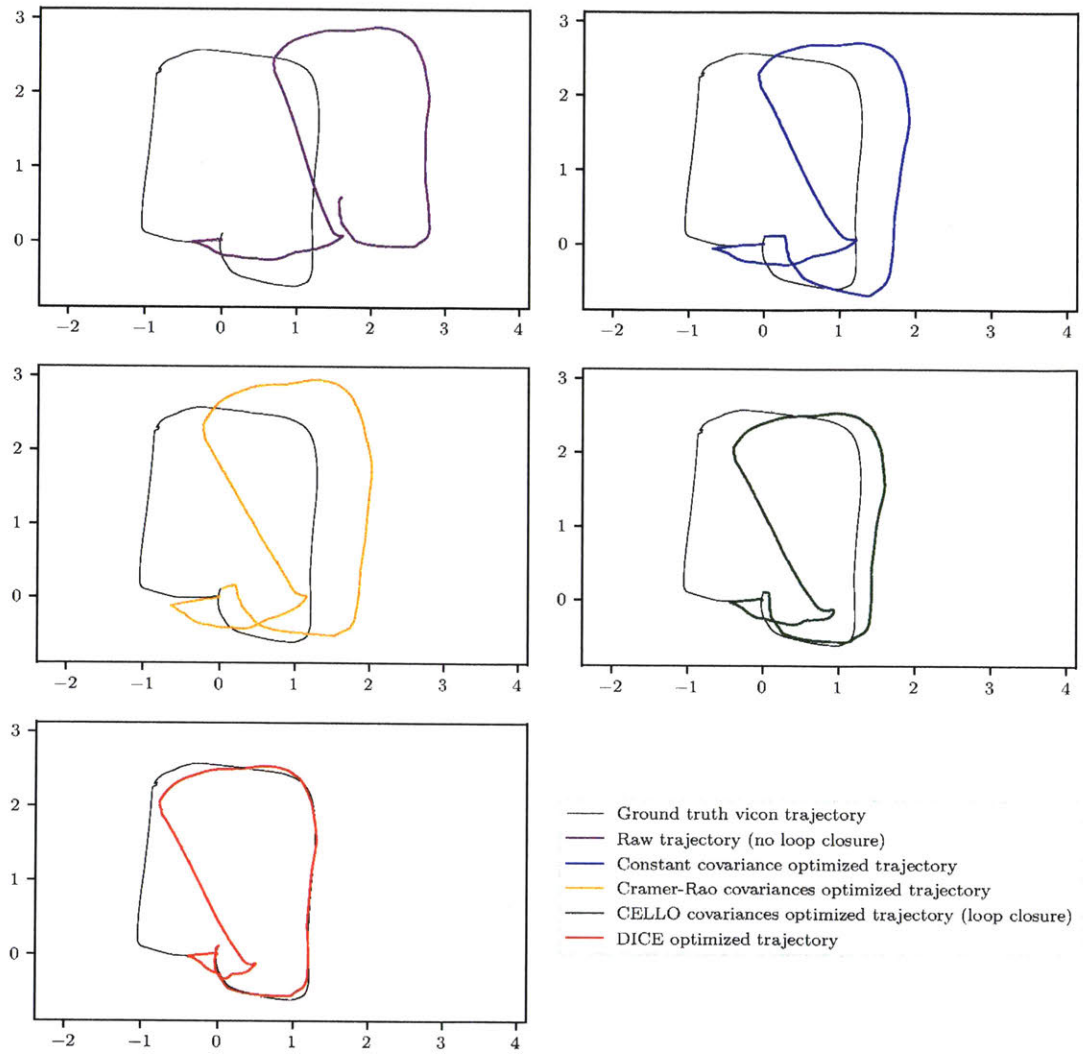


Figure 5-12: Trajectory tracking results for Dynamic test set.

However, for failure modes that are difficult to explain with the expert-specified features, DICE outperformed CELLO, demonstrating the utility of learning the feature space directly from the data.

In terms of the average rotational error, there was no significant difference between the covariance prediction methods. As shown in Fig. 5-15, the different methods were relatively accurate in estimating the orientation, and the empirically-fit constant covariances often performed the best. In the *White-wall* dataset, the constant covariance method outperformed DICE by 3.3 degrees, and this was the greatest difference

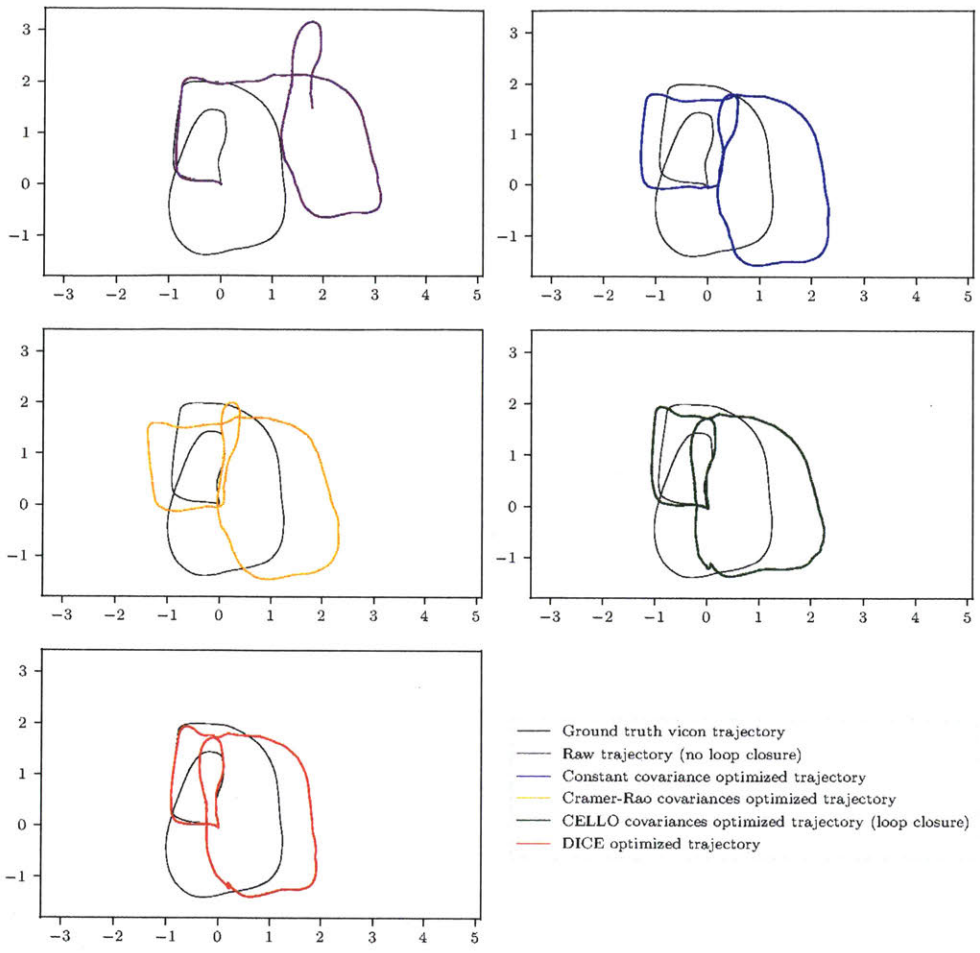


Figure 5-13: Trajectory tracking results for Dynamic-Unseen test set.

across all datasets. In conclusion, we observed that in cases where the raw estimates were accurate, there was negligible difference between DICE and the other methods, but in cases where the estimates were noisy, DICE out-performed other methods by a significant margin.

It is useful to comment on where the biggest gains can be made in terms of these loop closure experiments. Due to the fact that there is only one source of information, the most drastic improvements are evident when sensor measurement characterized by large covariances occur early in the trajectory; if a covariance prediction algorithm properly predicts a large covariance in this region, the later portion of the trajectory can be better reconstructed. However, small to negligible gains will generally be

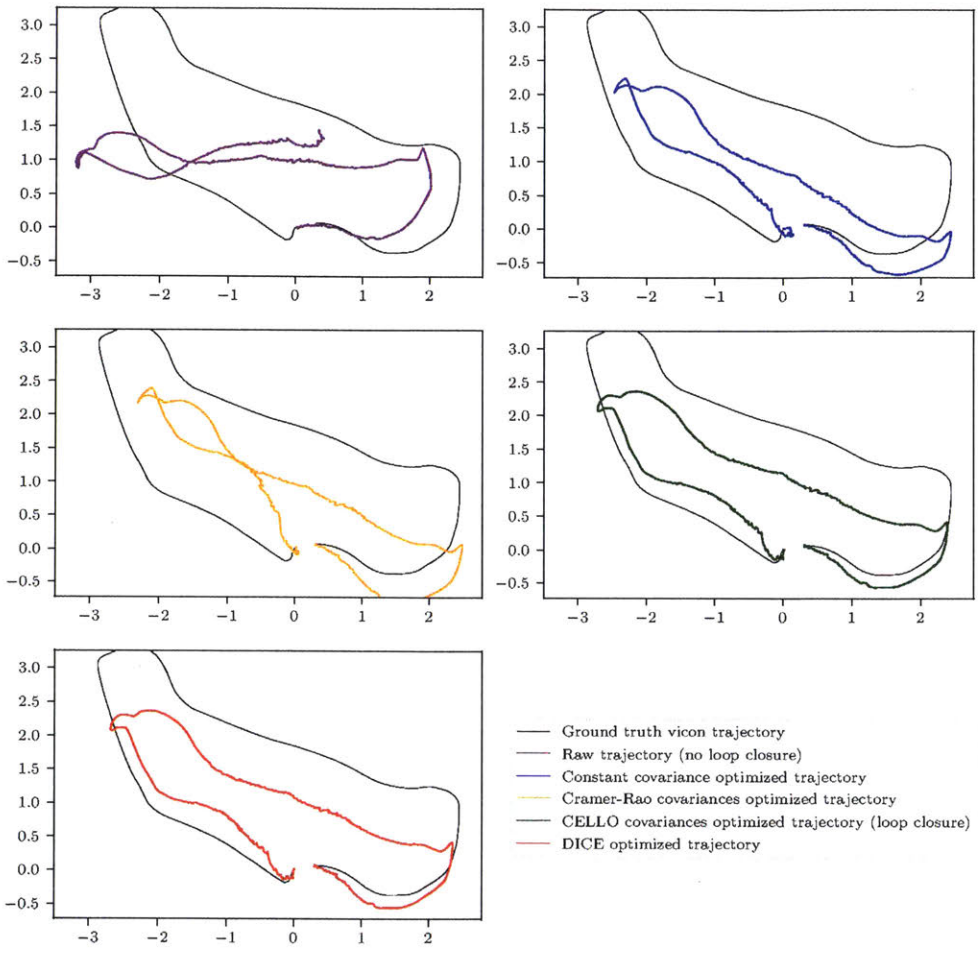


Figure 5-14: Trajectory tracking results for White-Wall test set.

made if the noisy measurements are concentrated at the end of the trajectory. This is because while we have a measure of uncertainty, the measurement itself may have too much error to correct. Were a second sensor available, improved covariance prediction could then allow for optimal sensor fusion over the entire trajectory. We have chosen test sets to highlight the benefits of improved covariance estimation in the absence of a second sensor.

We have shown that a deep convolutional neural network can be used to predict the covariance of Gaussian noise models for a RBG-D odometry algorithm. Covariances predicted by DICE outperform the constant covariance method, a scaled Cramer-Rao bound method, and an implementation of CELLO in terms of average log-likelihood

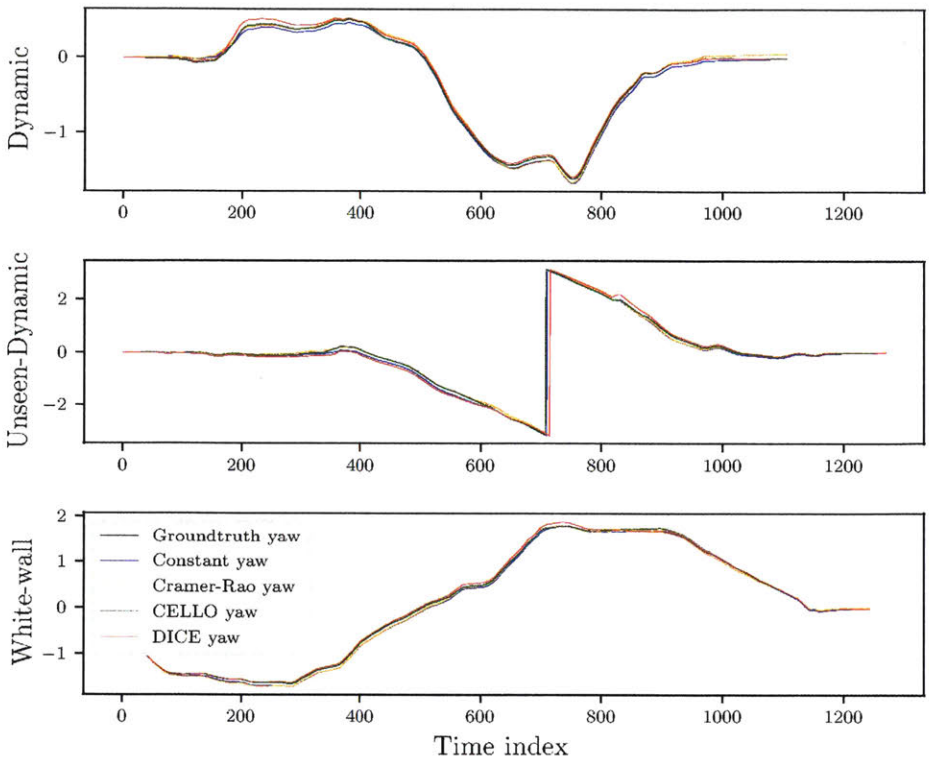


Figure 5-15: Comparison of orientation estimates (radians) using constant, Cramer-Rao, CELLO, and DICE measurement covariances. The orientation tracking of all methods is comparable and the constant covariances often performed the best.

over several test sets. Furthermore, we have demonstrated that using DICE predicted covariances can also improve loop closure performance.

Chapter 6

Conclusions and Future Work

We have presented DICE – a method for predicting measurement noise of complex sensors without using extensive domain knowledge. We demonstrated that DICE can accurately predict the measurement covariance of a simulated light sensor, and a visual odometry sensor. We have shown that predicting accurate measurement covariances can help improve trajectory estimates, and achieved accuracy significantly better than conventional methods for difficult scenes.

DICE is a two-stage algorithm. The first stage is the data collection and offline training stage. Raw sensor data and the accompanying error labels are collated into a dataset, which is then used as training data for the supervised training of a deep convolutional neural network. The network uses a deep neural network to predict the parameters of a modified Cholesky decomposition that best explains the observed data by minimizing a negative log-likelihood loss function. After the weights of the model have been trained, DICE can then be used for the second stage, i.e., online query. At query time, raw sensor measurements are passed through the model in the standard feed-forward formulation. The measurement covariance is then reconstructed from the output vector of the CNN.

Our approach presents several benefits over other approaches to predictive Gaussian noise models. Firstly, DICE is a flexible, algorithm-agnostic method that does not require hand-coded features to be specified by a domain expert. Instead, feature discovery is handled through the optimization of the model itself via the optimization

of convolutional layers. This allows for a more straight-forward and accessible deployment model. All that is required is a representative dataset of sensor measurements and errors.

Another benefit is that ground-truth covariance labels are not required to predict the noise parameters of the measurement models. It is often infeasible to obtain such labels (one approach might involve running many expensive simulations, and for some applications it is not straightforward to construct such a simulation). Instead, the network optimizes the overall likelihood over the observed error data, which is in general easier to collect.

Furthermore, DICE is computationally efficient; while offline training can be expensive due to the volume of data that must be processed, at run-time the feed-forward loop is fast enough for real-time filtering applications. Low computation times are a requirement for a predictive noise model to be relevant on real systems. Unlike non-parametric techniques, the CNN does not scale at run time with the amount of data. Unlike parametric techniques, the CNN does not sacrifice representational richness for computational efficiency.

Of course, using convolutional neural networks to predict noise parameters is not without some disadvantages. The burden of tuning the low dimensional feature representation has been shifted to network design; network structure, learning rates, etc., are design parameters that must be chosen. More hands-off network design is an area of active research (see Smith et al. [66] and Baker et al. [5], etc).

Additionally, while this method circumvents the need for ground truth covariance labels, the algorithm still requires groundtruth error labels as a training signal. Neural network approaches are notoriously data-hungry; coupled with the ground-truth requirement, this may cause some datasets to still be expensive to collect. A possible solution in these scenarios may be to leverage simulation environments; recent successes have shown simulated data to be useful for generating vast quantities of labeled data (see Shrivastava et al. [63] and Stein and Roy [68]).

Like all machine learning techniques, DICE can suffer from data imbalance issues. This can largely be mitigated by ensuring that the run-time queries are drawn from

the same distribution as the training data. It may ultimately be useful to perform novelty detection. Given some metric on the novelty of the input data, the state estimation algorithms can fall back to conservative estimates when the query appears to be novel by some measure.

While the potential advantages of using DICE for state estimation have been shown in this work, we observe that there are exciting applications in the planning domain as well. Given predictive noise models for complex sensors, uncertainty-aware planning techniques could be leveraged to plan trajectories that optimize tracking performance (see Prentice and Roy [56]). For example, if visual odometry is used to localize in known maps or meshes (see Ok et al. [52]), the learned measurement covariance could be used to minimize tracking error by avoiding paths that result in low-texture sensor readings.

A useful extension to the applications presented in this work would be to extend the frame-to-frame formulation into a frame-to-keyframe formulation; i.e., instead of predicting the uncertainty in the odometry between two frames, predict the uncertainty between two arbitrary frames. This would allow for the learned covariances to be used in keyframe-based SLAM techniques; not only could more constrained graphs be constructed to reduce drift, the measurement covariance is a natural metric for keyframe selection.

Beyond keyframe selection, a frame-to-keyframe measurement covariance could be used to enforce graph sparsity in the graph based optimization techniques discussed in this work. Instead of keeping all dense measurements, which can lead to longer solve times, with informative and accurate covariances, only those factors which contain the least uncertainty could be maintained.

Bibliography

[1] Amazon Prime Air. https://www.amazon.com/Amazon-Prime-Air/b?node=8037720011. Accessed Dec 18 2017.

[2] Kinect sensor main page. https://msdn.microsoft.com/en-us/library/hh438998.aspx. Accessed Dec 18 2017.

[3] Zoom in, Zoom out: Speedy, Agile UAVs Envisioned for Troops in Urban Missions. https://www.darpa.mil/news-events/2014-12-22, 2000. Accessed Dec 18 2017.

[4] Mohammad RN Avanaki, P Philippe Laissue, Tae Joong Eom, Adrian G Podoleanu, and Ali Hojjatoleslami. Speckle reduction using an artificial neural network algorithm. *Applied optics*, 52(21):5050–5057, 2013.

[5] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[6] Martin Barczyk and Silvère Bonnabel. Towards realistic covariance estimation of ICP-based Kinect V1 scan matching: The 1D case. In *American Control Conference (ACC), 2017*, pages 4833–4838. IEEE, 2017.

[7] Martin Barczyk, Silvere Bonnabel, and François Goulette. Observability, covariance and uncertainty of icp scan matching, 2014.

[8] Gavin C Cawley, Gareth J Janacek, Malcolm R Haylock, and Stephen R Dorling. Predictive uncertainty in environmental modelling. *Neural networks*, 20(4):537–549, 2007.

[9] Andrea Censi. An accurate closed-form estimate of ICP's covariance. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 3167–3172. IEEE, 2007.

[10] Harald Cramér. *Mathematical Methods of Statistics (PMS-9)*, volume 9. Princeton university press, 2016.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[12] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

[13] Alex Davies. Google's Self-Driving Car Dream Is Finally Coming Real. `https://www.wired.com/story/waymo-google-arizona-phoenix-driverless-self-driving-cars`, Nov 2017. Accessed Dec 18 2017.

[14] Sander Dieleman et al. Lasagne: First release., August 2015. URL `http://dx.doi.org/10.5281/zenodo.27878`.

[15] D'Orazio, Dante. Point Grey Flea3 camera captures 4K video in a 1.2-inch cube form factor for $949. `https://www.theverge.com/2012/7/1/3128515/point-grey-flea3-worlds-smallest-4k-video-camera`, Jul 2012.

[16] Arnaud Doucet, Nando De Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 176–183. Morgan Kaufmann Publishers Inc., 2000.

[17] Jindřich Duník, Ondřej Straka, Oliver Kost, and Jindřich Havlík. Noise covariance matrices in state-space models: A survey and comparison of estimation methodsPart I. *International Journal of Adaptive Control and Signal Processing*, 2017.

[18] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proc. ECCV 2014*, pages 834–849. Springer, 2014.

[19] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Proc. ICRA 2014*, pages 15–22. IEEE, 2014.

[20] Samuel Gibbs. Uber signs contract with NASA to develop flying taxi software. `https://www.theguardian.com/technology/2017/nov/08/uber-signs-contract-nasa-develop-flying-taxi-software`, Nov 2017. Accessed Dec 18 2017.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

[23] H Heffes. The effect of erroneous models on the Kalman filter response. *IEEE Transactions on Automatic Control*, 11(3):541–543, 1966.

[24] Ralf Herbrich, Neil D Lawrence, and Matthias Seeger. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in neural information processing systems*, pages 625–632, 2003.

[25] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[27] Humphrey Hu and George Kantor. Parametric covariance prediction for heteroscedastic noise. In *Proc. IROS 2015*, pages 3052–3057. IEEE, 2015.

[28] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[29] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[30] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[31] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for RGB-D cameras. In *Proc. ICRA 2013*, pages 3748–3754. IEEE, 2013.

[32] Jonathan Ko and Dieter Fox. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[34] Věra Kůrková. Kolmogorov's theorem and multilayer neural networks. *Neural networks*, 5(3):501–506, 1992.

[35] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[36] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[37] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[38] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, 1997.

[39] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

[40] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[41] D Magill. Optimal adaptive estimation of sampled stochastic processes. *IEEE Transactions on Automatic Control*, 10(4):434–439, 1965.

[42] Peter S Maybeck and Donald L Pogoda. Multiple model adaptive controller for the stol f-15 with sensor/actuator failures. In *Decision and Control, 1989., Proceedings of the 28th IEEE Conference on*, pages 1566–1572. IEEE, 1989.

[43] Raman Mehra. On the identification of variances and adaptive Kalman filtering. *IEEE Transactions on automatic control*, 15(2):175–184, 1970.

[44] Timothy E Menke and Peter S Maybeck. Sensor/actuator failure detection in the Vista F-16 by multiple model adaptive estimation. *IEEE Transactions on aerospace and electronic systems*, 31(4):1218–1229, 1995.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[46] Dharmendra S Modha and Yeshaiahu Fainman. A learning law for density estimation. *IEEE Transactions on Neural Networks*, 5(3):519–523, 1994.

[47] AH Mohamed and KP Schwarz. Adaptive Kalman filtering for INS/GPS. *Journal of geodesy*, 73(4):193–203, 1999.

[48] Elizbar A Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[49] Yurii Nesterov. A method of solving a convex programming problem with convergence rate O (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[50] T Nishimura. Error bounds of continuous Kalman filters and the application to orbit determination problems. *IEEE transactions on automatic control*, 12(3): 268–275, 1967.

[51] Giuseppe Nunnari, Stephen Dorling, Uwe Schlink, Gavin Cawley, Rob Foxall, and Tim Chatterton. Modelling SO 2 concentration at a point with statistical approaches. *Environmental Modelling & Software*, 19(10):887–905, 2004.

[52] Kyel Ok, W Nicholas Greene, and Nicholas Roy. Simultaneous tracking and rendering: Real-time monocular localization for MAVs. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4522–4529. IEEE, 2016.

[53] Valentin Peretroukhin, Lee Clement, Matthew Giamou, and Jonathan Kelly. PROBE: Predictive robust estimation for visual-inertial navigation. In *Proc. IROS 2015*, pages 3668–3675. IEEE, 2015.

[54] Valentin Peretroukhin, William Vega-Brown, Nicholas Roy, and Jonathan Kelly. PROBE-GK: Predictive robust estimation using generalized kernels. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 817–824. IEEE, 2016.

[55] Sai Manoj Prakhya, Liu Bingbing, Yan Rui, and Weisi Lin. A closed-form estimate of 3D ICP covariance. In *Machine Vision Applications (MVA), 2015 14th IAPR International Conference on*, pages 526–529. IEEE, 2015.

[56] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

[57] Joaquin Quiñonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

[58] C Radhakrishna Rao. Information and the accuracy attainable in the estimation of statistical parameters. In *Breakthroughs in statistics*, pages 235–247. Springer, 1992.

[59] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[60] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[61] Amir Sarajedini, Robert Hecht-Nielsen, and Paul M Chau. Conditional probability density function estimation with sigmoidal neural networks. *IEEE Transactions on neural networks*, 10(2):231–238, 1999.

[62] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[63] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from simulated and unsupervised images through adversarial training. *arXiv preprint arXiv:1612.07828*, 2016.

[64] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[66] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer, 1990.

[67] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[68] Gregory J Stein and Nicholas Roy. GeneSIS-RT: Generating Synthetic Images for training Secondary Real-world Tasks. *arXiv preprint arXiv:1710.04280*, 2017.

[69] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Proc. IROS*, Oct. 2012.

[70] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[71] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[72] William Vega-Brown. Predictive parameter estimation for Bayesian filtering. Master's thesis, Massachusetts Institute of Technology, 2013.

[73] William Vega-Brown, Abraham Bachrach, Adam Bry, Jonathan Kelly, and Nicholas Roy. CELLO: A fast algorithm for covariance estimation. In *Proc. ICRA 2013*, pages 3160–3167. IEEE, 2013.

[74] Halbert White. Parametric statistical estimation with artificial neural networks: A condensed discussion. *From statistics to neural networks: theory and pattern recognition applications*, 136:127, 1994.

[75] Peter M Williams. Using neural networks to model conditional multivariate densities. *Neural Computation*, 8(4):843–854, 1996.

[76] Peter M Williams. Matrix logarithm parametrizations for neural network covariance models. *Neural Networks*, 12(2):299–308, 1999.

[77] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[78] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE Multimedia Mag.,*, 19(2):4–10, 2012.